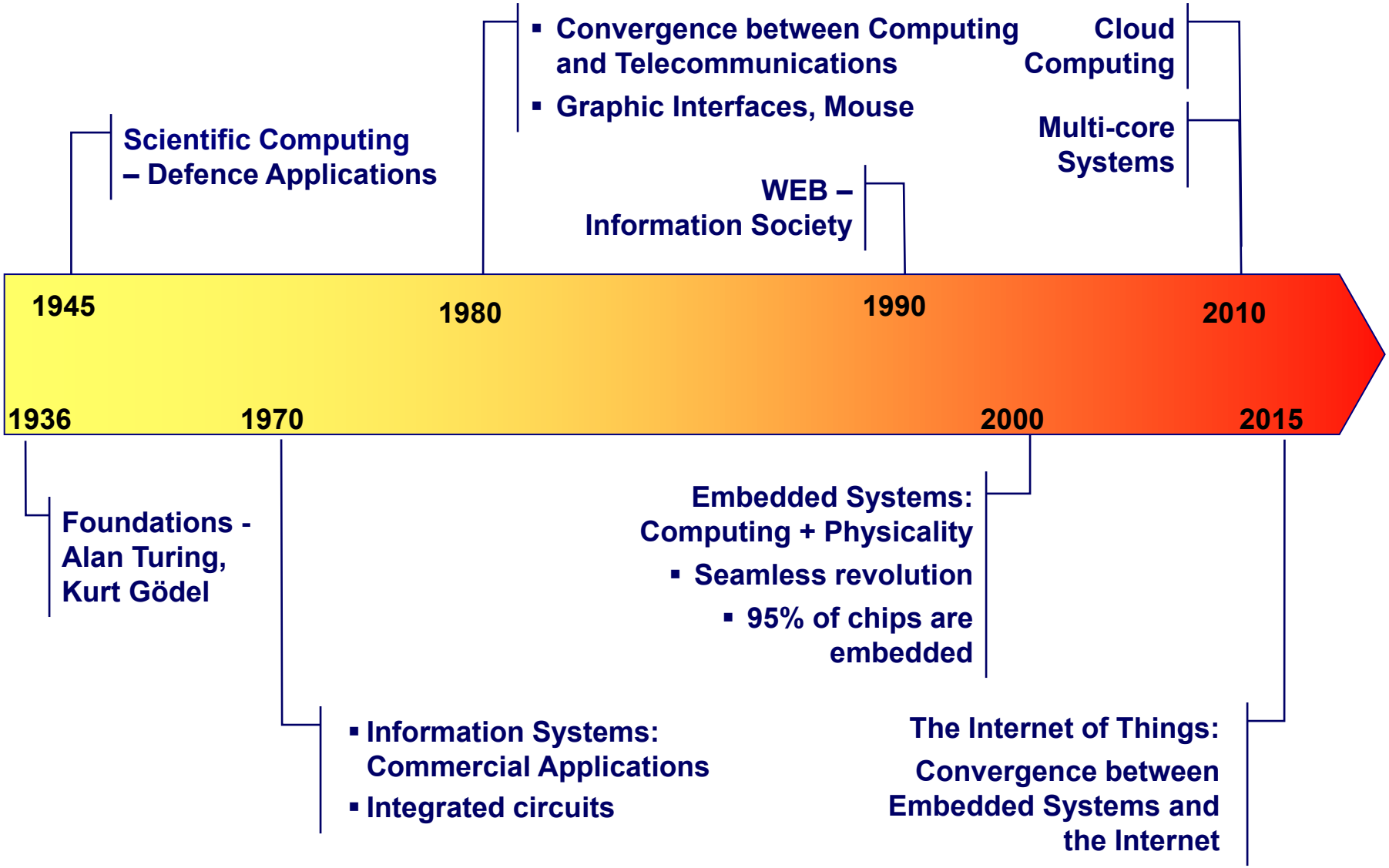
Rigorous System Design in BIP

Nano-Tera/Artist Summer School on Embedded System Design

Aix-les-Bains, September 17 - 21, 2012

Joseph Sifakis
EPFL and Verimag Laboratory
in collaboration with
A. Basu, S. Bensalem, S. Bliudze, M. Bozga, J. Combaz, H. Nguyen, M. Jaber, J. Quilbeuf

From Programs to Systems – The Evolution of IST



Evolution driven by exponential progress in technology and explosion of applications

Reactive Systems – The Hardest and the Most Important

Shift of focus from transformational software to reactive systems!

Reactive System
(SW+HW)

Reactive Systems – The Hardest and the Most Important

Significant differences

Transformational Software
- terminating
- deterministic
- platform-independent
- Theory

Reactive System
- non-terminating
- non-predictable
- platform-dependent
- No theory!

Reactive systems

- are hard to design due to unpredictable and subtle interactions with the environment, emergent behaviors, and occasional catastrophic cascading failures rather than to complex data and algorithms
- are increasingly important in modern computing systems: embedded systems, cyber-physical systems, mobile systems, web-services

System Design – An Increasing Gap

System Design – Multicore systems

Gartner, Research Note [09]: *"Software is struggling to keep pace with the fast growth of multicore processors" … "Running advanced multicore machines with today's software is like "putting a Ferrari engine in a go-cart,"… "Many of the software configurations in use today will be challenged to support the hardware configurations possible, and those will be accelerating in the future."*

Intel [06]: *"Multi-core processing is taking the industry on a fast moving and exciting ride into profoundly new territory. The defining paradigm in computing performance has shifted inexorably from raw clock speed to parallel operations and energy efficiency."*

Microsoft Research [07]: *"Multicore processors represent one of the largest technology transitions in the computing industry today, with deep implications for how we develop software."*

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

System Design – About Design

Design is a universal concept,
a par excellence intellectual activity
leading to artifacts meeting given requirements.

Easy Apple Pie

Proceduralization

RECIPE
(Program)

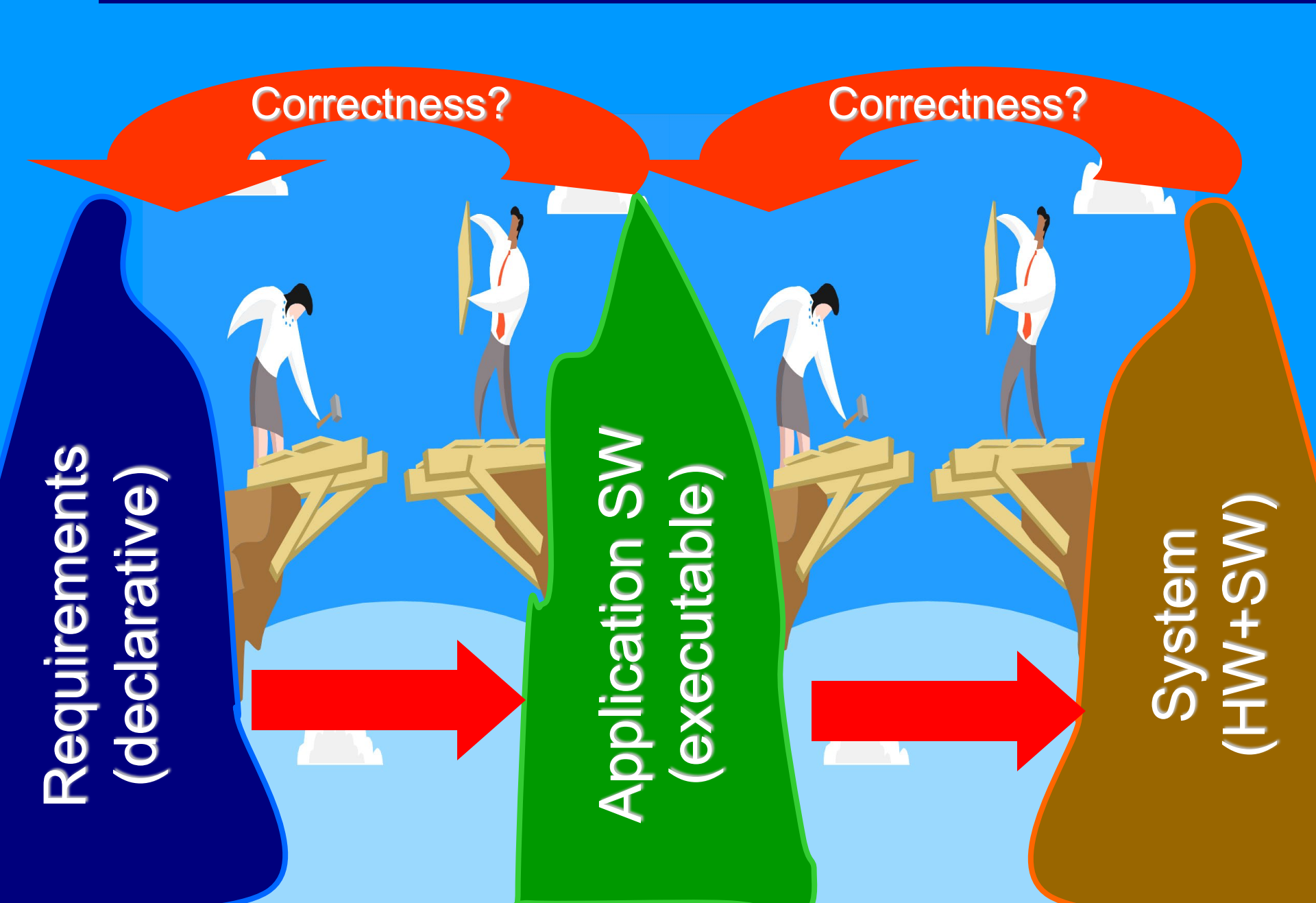
- Put apples in pie plate;
- Sprinkle with cinnamon and 1 tablespoon sugar;
- In a bowl mix 1 cup sugar, flour and butter;
- Blend in unbeaten egg, pinch of salt and the nuts;
- Mix well and pour over apples;
- Bake at 350 degrees for 45 minutes

Materialization

INGREDIENTS
(Resources)

- 1 pie plate buttered
- 5or 6 apples, cut up
- ¾ c. butter, melted
- 1 c. flour
- ½ c. chopped nuts
- 1tsp cinnamon
- 1tbsp sugar
- 1c. Sugar

System Design – Two Main Gaps



System Design – Productivity vs. Correctness

System designers strive to reconcile two often conflicting demands:

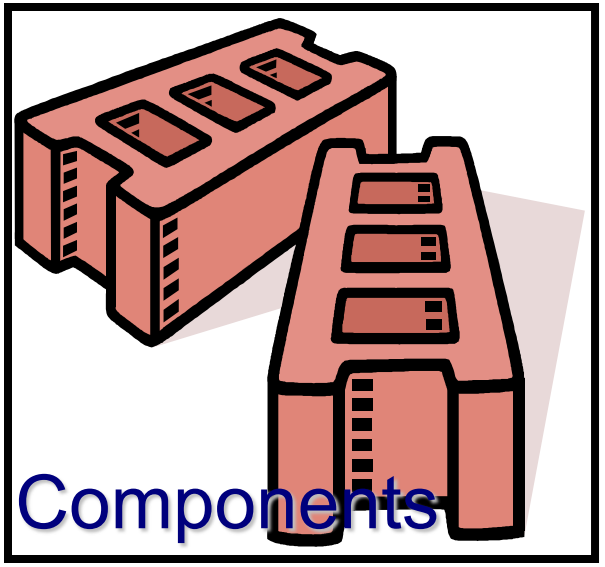
- Productivity characterizes the efficiency of the design process.
- Correctness means compliance to requirements

As flawless system design is not attainable, owing to both theoretical limitations and cost-effectiveness considerations, system designers target levels of criticality.

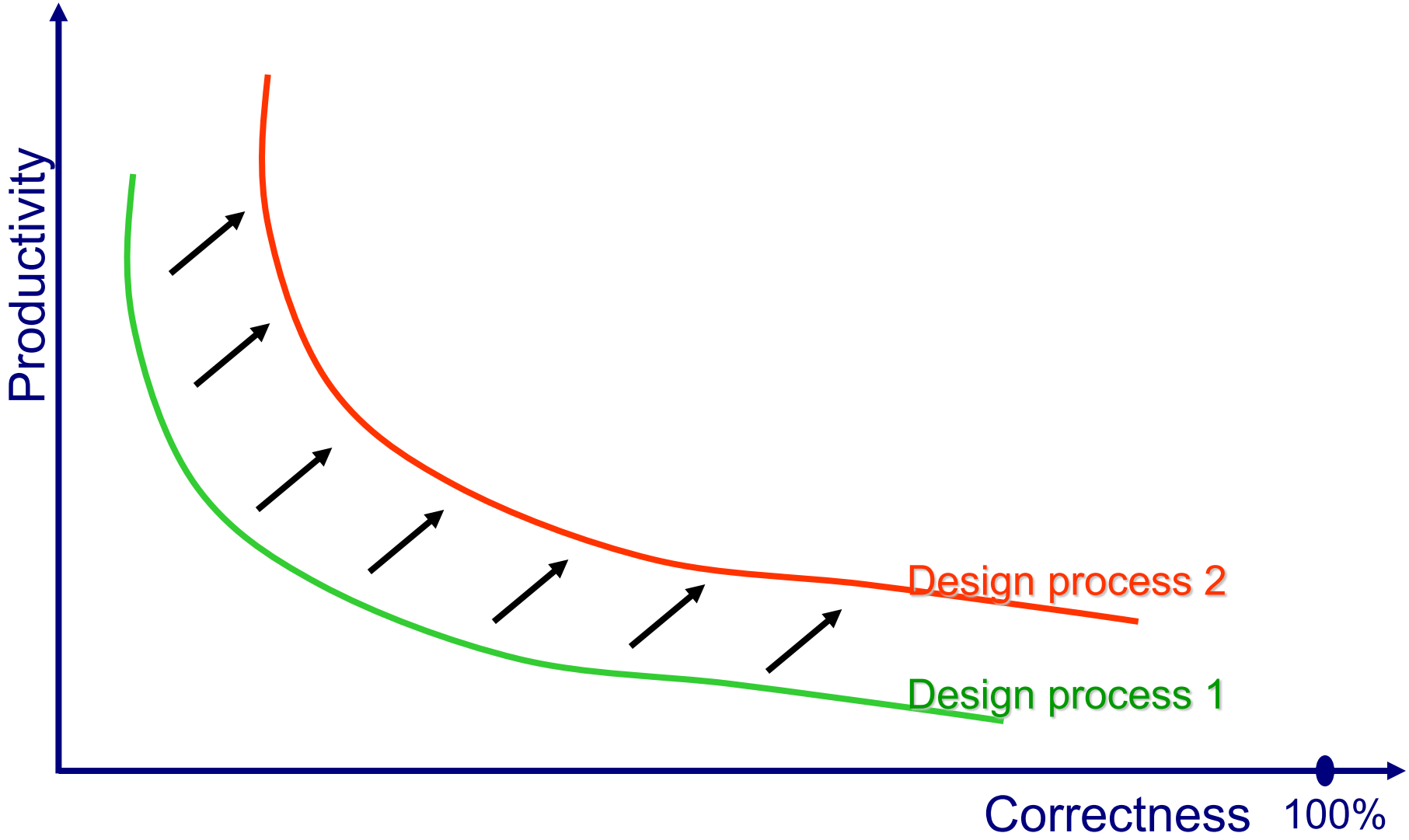
Levels of criticality

- Correspond to tradeoffs between correctness and productivity.
- Determine which types of requirements are relevant and to what extent these requirements should be met e.g. probability of failure or disparity between nominal and observed values of significant parameters.

Efficiency of the design process



System Design – Productivity vs. Correctness



System Design – Correctness

Trustworthiness requirements express assurance that the designed system can be trusted that it will perform as expected despite

HW failures

Design Errors

Environment Disturbances

Malevolent Actions

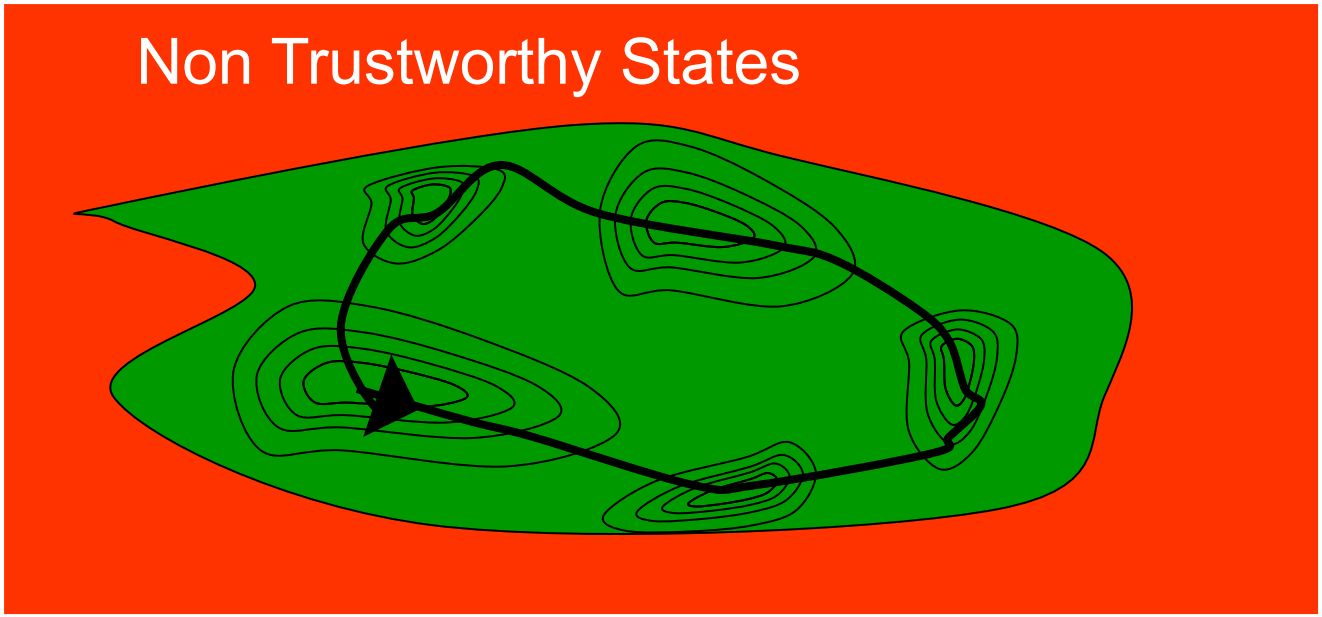
Optimization requirements dealing with optimization of functions subject to quantitative constraints on

1) Performance: how well the system does wrt user demands e.g. throughput, jitter, latency, quality.
2) Cost: how well resources are used wrt economic demands e.g. storage efficiency, processor utilizability, energy efficiency.

Usually they determine tradeoffs between performance and cost

System Design – Trustworthiness vs. Optimization

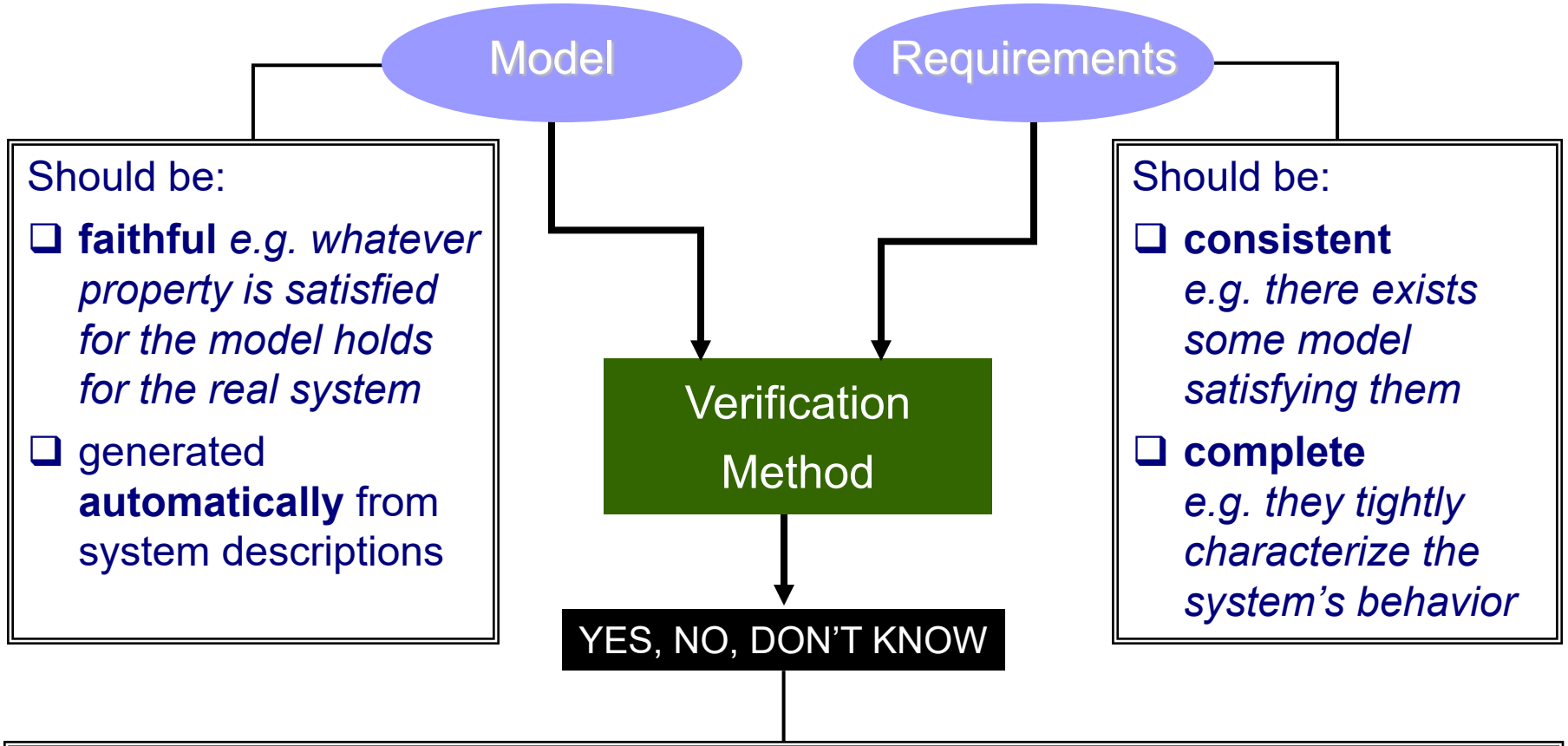
- Trustworthiness requirements characterize qualitative correctness – a state is either trustworthy or not
- Optimization requirements characterize execution sequences



Trustworthiness vs. Optimization

The two types of requirements are often antagonistic: trustworthy designs give rise to non optimized solutions and conversley

System Design – Correctness-by-checking



- As a rule, for infinite state models all non trivial properties are undecidable e.g. x<100
- Intrinsically high complexity for finite state models (state explosion problem)

System Design – Correctness-by-Checking: Limitations

- It is a relative judgment "*Are we building the system right*?" It would be an answer to the question "*Are we building the right system*?" if

  1. requirements could be correctly formalized, sound and complete

  Effective use of rigorous requirement specification languages for real-life systems is problematic

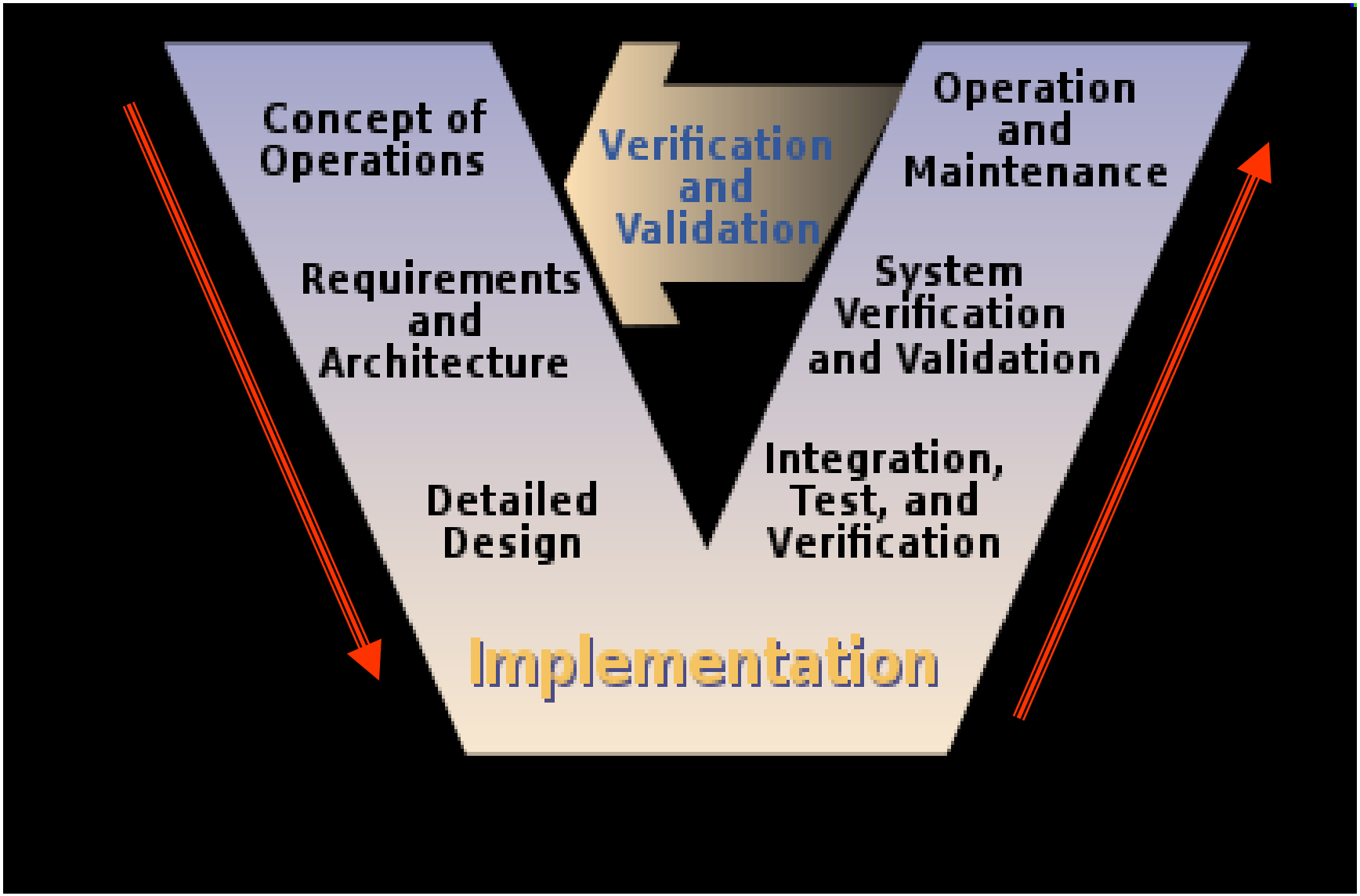
  2. system models could faithfully represent the system behavior interacting with its environment

  Generating models even for very simple systems, such as the node of a wireless sensor network, requires understanding intricate interaction between application software and the underlying execution platform

- Contributes to trustworthiness but it is restricted to requirements that can be formalized and checked efficiently

- For optimization requirements, a more natural approach for their satisfaction is by enforcing or synthesis rather than by checking

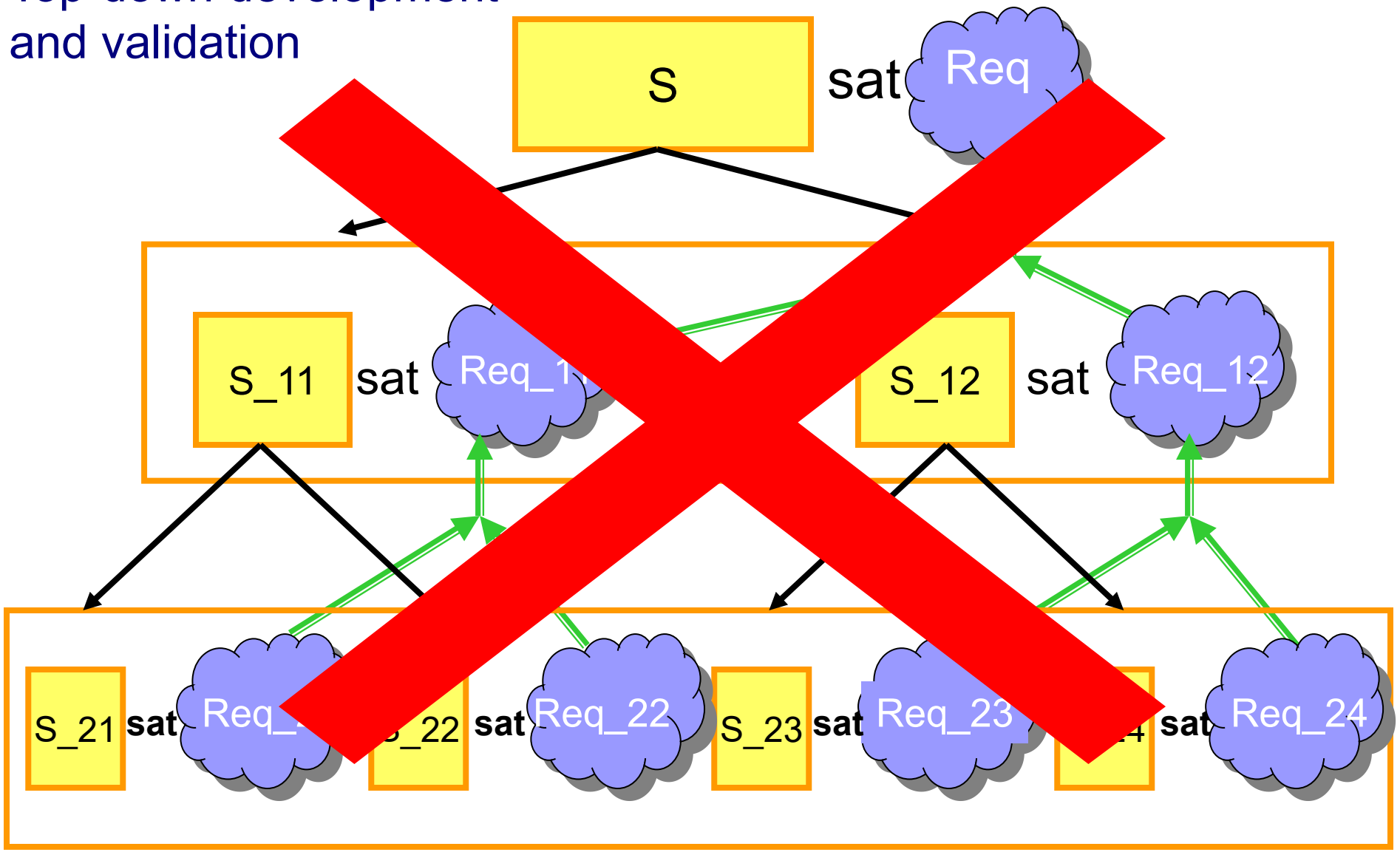
System Design – The V-model



The V-model of the Systems Engineering Process, Source: Wikipedia

System Design – The V-model

Top-down development and validation



System Design – The V-model

The V-model of the traditional Systems Engineering process

1. assumes that all the system requirements are initially known, can be clearly formulated and understood.

2. assumes that system development is top-down from a set of requirements. Nonetheless, systems are never designed from scratch; they are built by incrementally modifying existing systems and component reuse.

3. considers that global system requirements can be broken down into requirements satisfied by system components. Furthermore, it implicitly assumes a compositionality principle: if components are proven correct with respect to their individual requirements, then correctness of the whole system can be inferred from correctness of its components.

4. relies mainly on correctness-by-checking (verification or testing)

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Rigorous System Design – The Concept

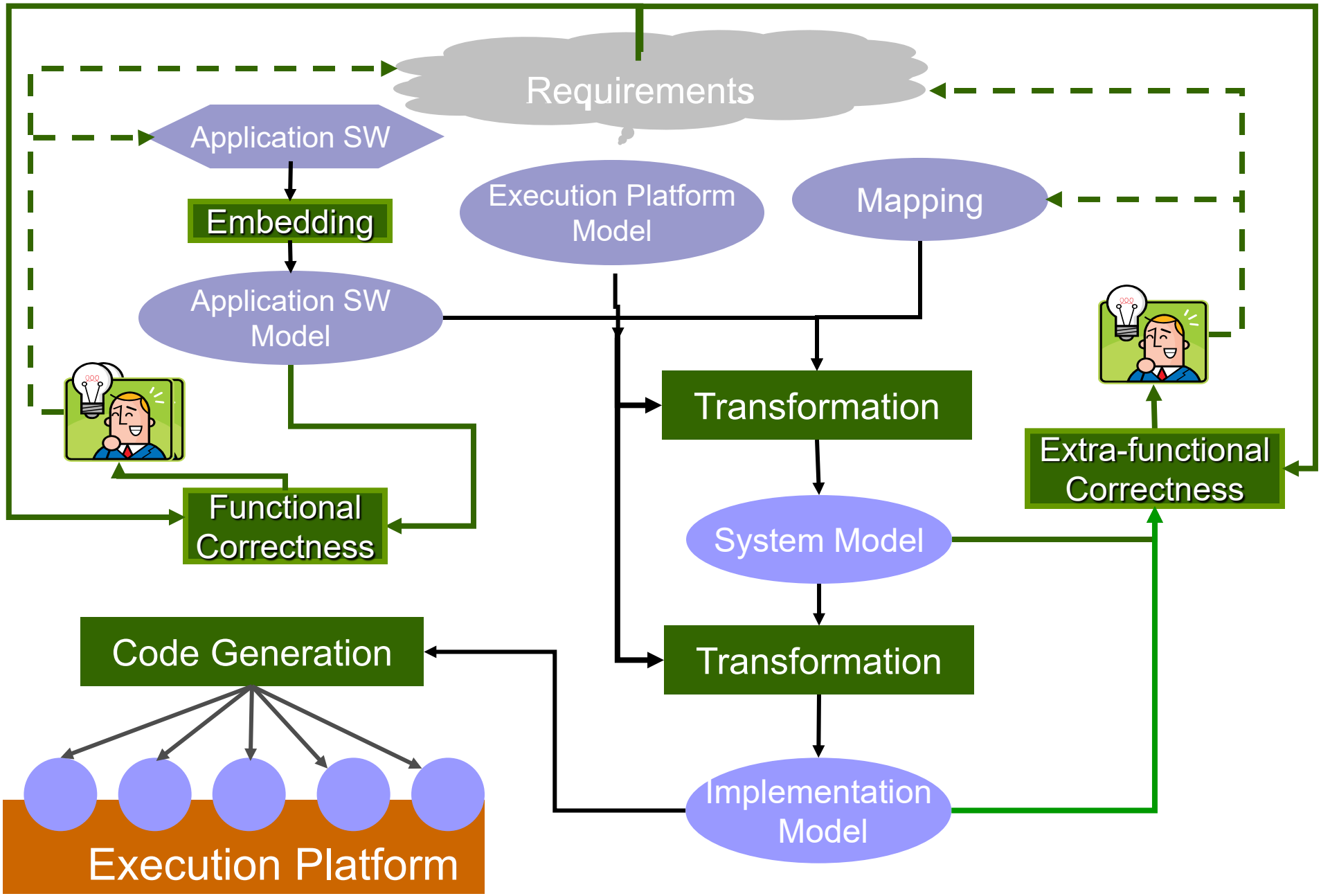
RSD considers design as a formal accountable and iterative process for deriving trustworthy and optimized implementations from an application software and models of its execution platform and its external environment

- Model-based: successive system descriptions are obtained by correct-by-construction source-to-source transformations of a single expressive model rooted in well-defined semantics
- Accountable: possibility to assert which among the requirements are satisfied and which may not be satisfied – accountability can be enhanced by using property-preservation results

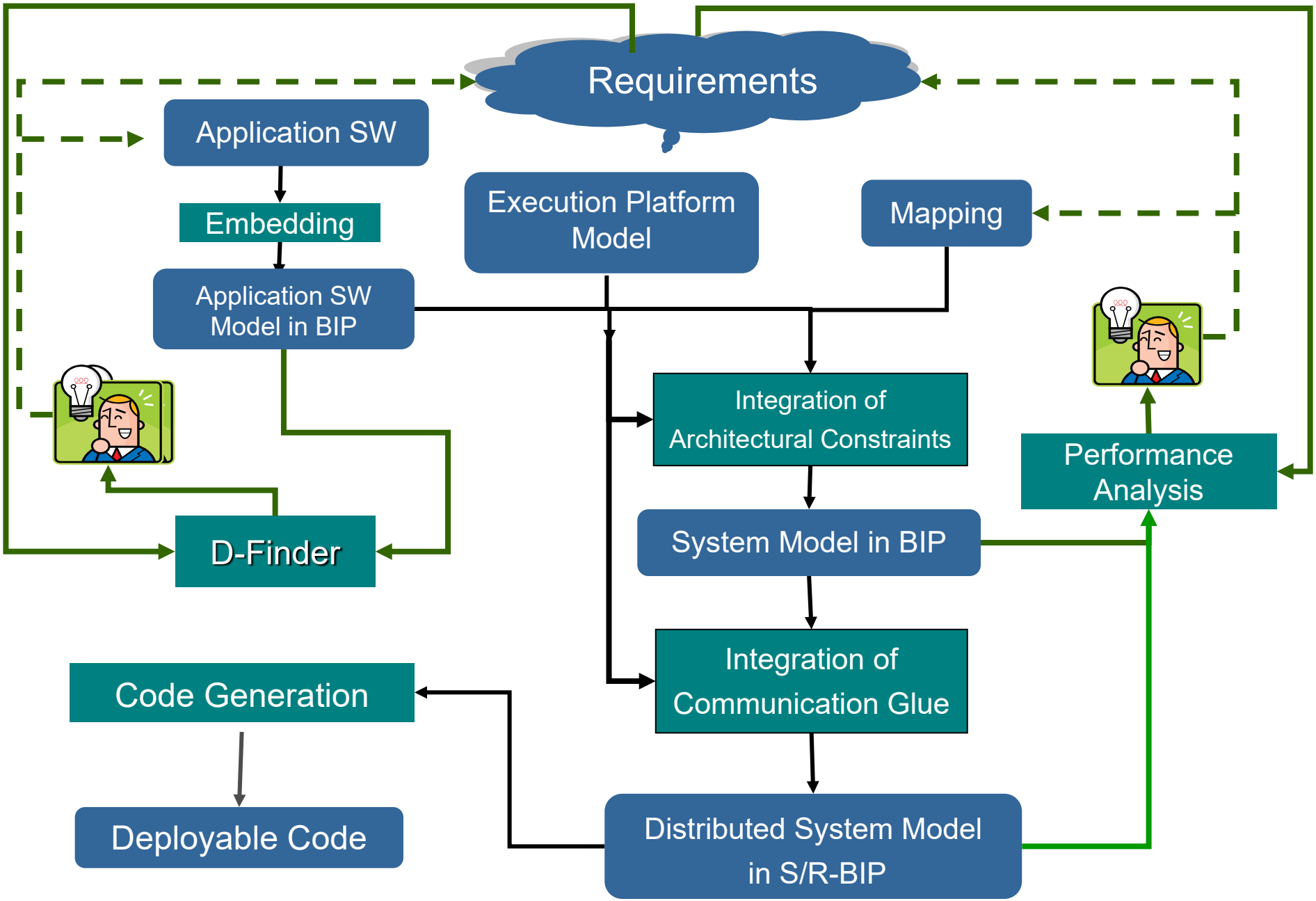
RSD focuses on mastering and understanding design as a process based on divide-and-conquer strategies involving iteration on a set of steps and clearly identifying

- points where human intervention and ingenuity are needed to resolve design choices through requirements analysis and confrontation with experimental results.
- segments of the design process that can be supported by tools to automate tedious and error-prone tasks.

Rigorous System Design – Simplified Flow



Rigorous System Design – Simplified Flow



Rigorous System Design – Is it attainable?

We should learn from two successful rigorous design paradigms:

- VLSI design and associated EDA tools have enabled the IC industry to sustain almost four orders of magnitude in product complexity growth since the 80386, while maintaining a consistent product development timeline.
- Safety-critical systems ensure trustworthy control of aircraft, cars, plants, medical devices

Main reasons of success

- Coherent and accountable design flows, supported by tools and often enforced by standards
- Correct-by-construction design enabled by extensive use of architectures and formal design rules

These are only instructive templates

- ICs consist of a limited number of fairly homogeneous components
- critical systems development techniques are not cost-effective for general purpose systems

Rigorous System Design – Principles

We need to study system design as a formal process leading from requirements to implementations.

Separation of concerns: Keep separate what functionality is provided (application SW) from how its is implemented by using resources of the target platform

Components: Use components for productivity and enhanced correctness

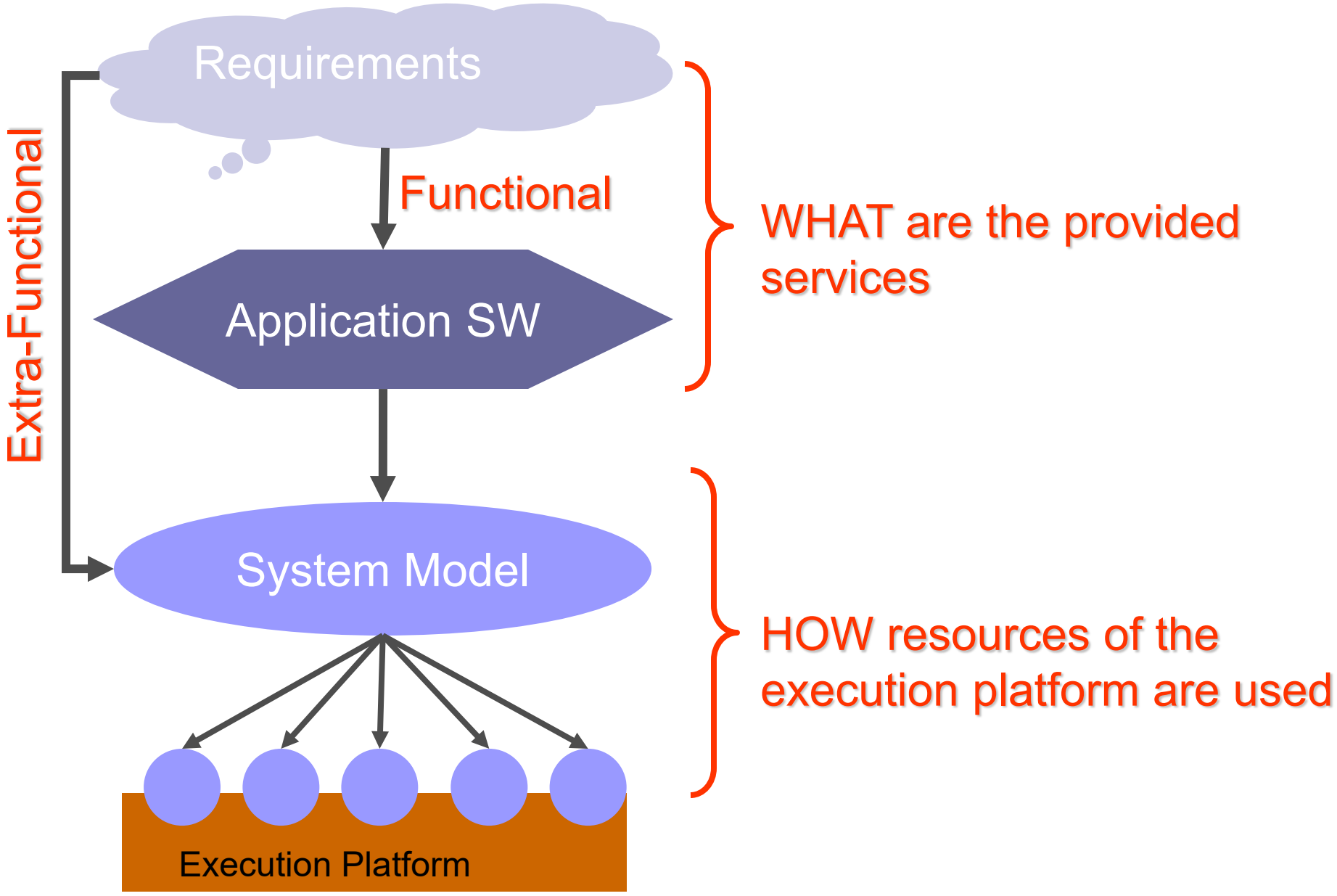
Coherency: Based on a single model to avoid gaps between steps due to the use of semantically unrelated formalisms e.g. for programming, HW description, validation and simulation, breaking continuity of the design flow and jeopardizing its coherency

Correctness-by-construction: Overcome limitations of a posteriori verification through extensive use of provably correct reference architectures and structuring principles enforcing essential properties

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Separation of Concerns



Separation of Concerns – SW vs. System Model

Application SW

Time and resources are external parameters that are linked to corresponding physical quantities of the execution environment

?

System Model

- Time and resources are state variables
- Each action consumes and liberates an amount of resources explicitly specified (resource parameters)
- Resource-Consistency
- Resource-Robustness

Separation of Concerns – Building a System Model

Resource-Consistency: Incremental and parallel modification of resources in a model should agree with laws governing physical resources.

Significant difference between model and physical time:
Physical time

- Is monotonically increasing.
- Its progress cannot be blocked

Model time

- can block or can involve Zeno runs (as in timed automata)
- Deadline miss = deadlock or time-lock.

Additional difficulties arise in resource modeling for models of distributed systems in particular because time is a global variable on which depend resource dynamics.

Separation of Concerns – Building a System Model

System analysis techniques assume resource-robustness: small change of resource parameters entail commensurable change of performance
Unfortunately, systems are not robust, in general.

For example, one would expect that a system model would exhibit worst performance for worst-case execution times of its actions.

- Performance degradation can be observed for increasing speed of the execution platform – Timing Anomaly

- Non determinism is one of the identified causes of such counter-intuitive behavior

We lack theory for guaranteeing resource-robustness

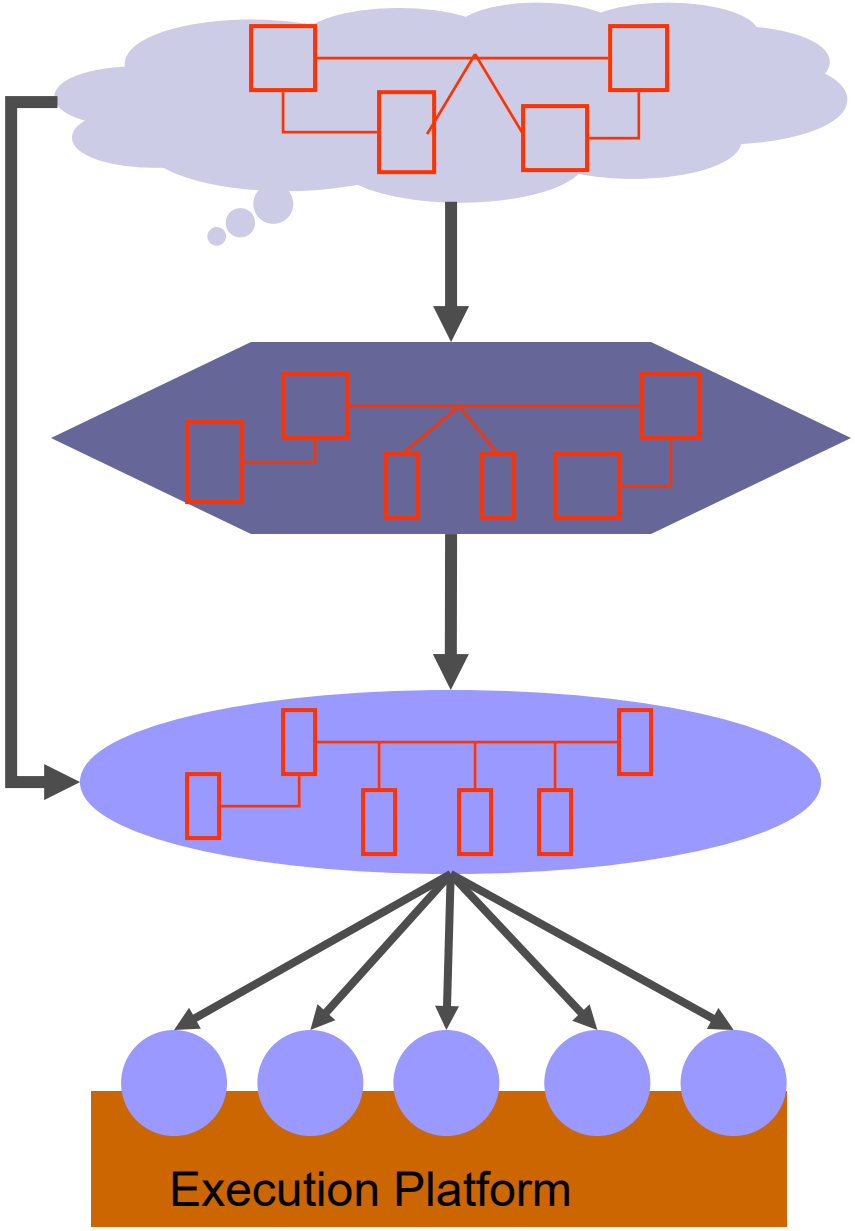
- performance should change monotonically with resources

- analysis for worst-case and best-case values of resource parameters suffice to determine performance bounds.

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Component-based Design



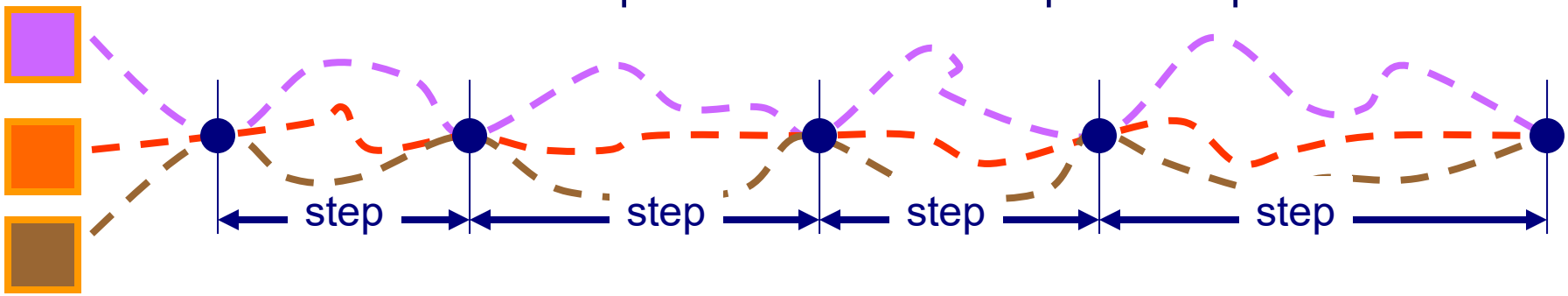
- Components are indispensable for enhanced productivity and correctness
- Component composition lies at the heart of the parallel computing challenge
- There is no Common Component Model - Heterogeneity

Component-based Design – Heterogeneity

Heterogeneity of Execution Modes

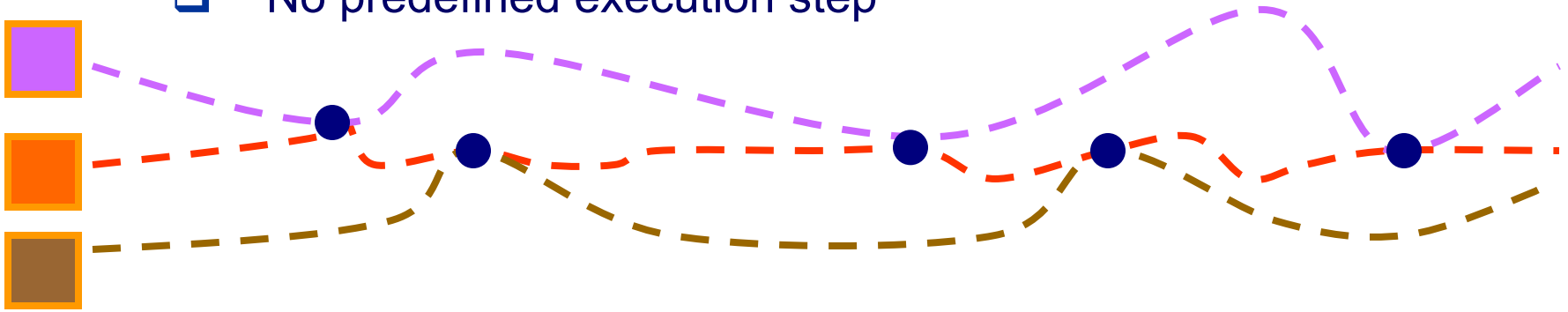
Synchronous components (HW, Multimedia application SW)

- Execution is a sequence of non interruptible steps



Asynchronous components (General purpose application SW)

- No predefined execution step



Open problem: Theory for consistently composing synchronous and asynchronous components e.g. GALS

Component-based Design – Heterogeneity

Synchronous Systems

Y(t+1) = f(Y(t),X(t))

dY/dt = f(Y(t),X(t))

Component: transfer function

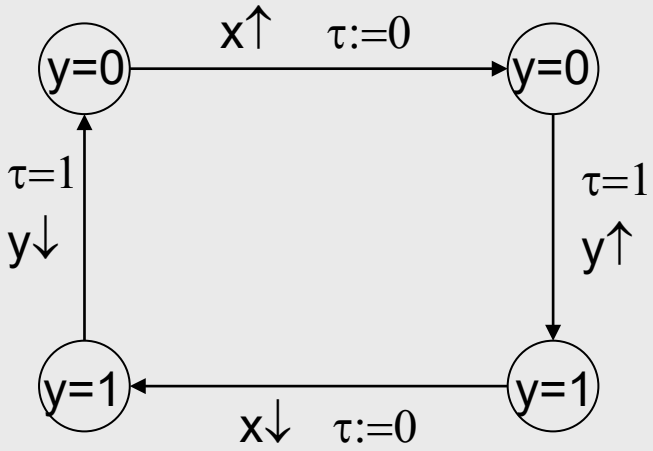
Composition: flow equalization

Asynchronous Systems

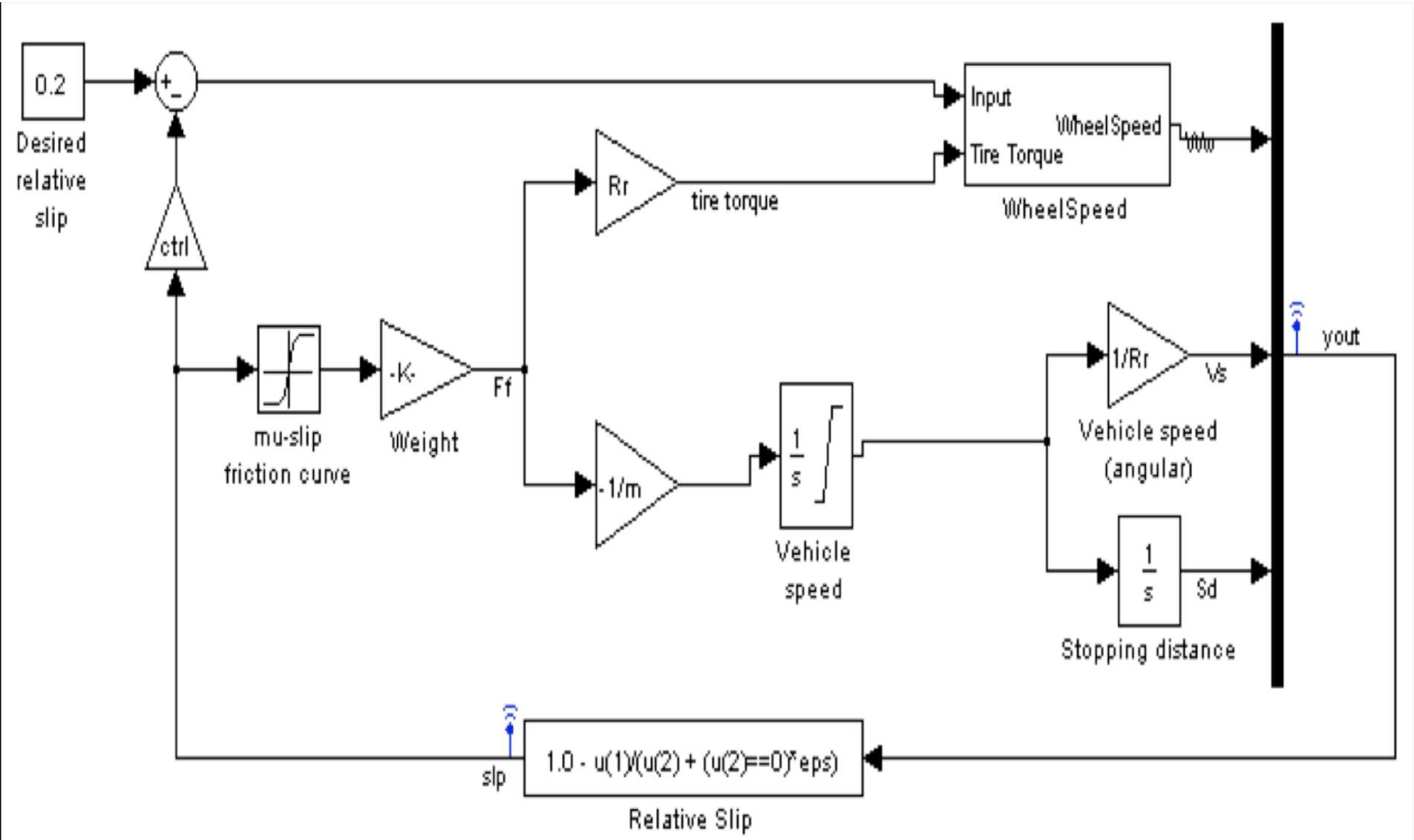
Interacting transition systems

Component: Transition system

Composition: Interaction



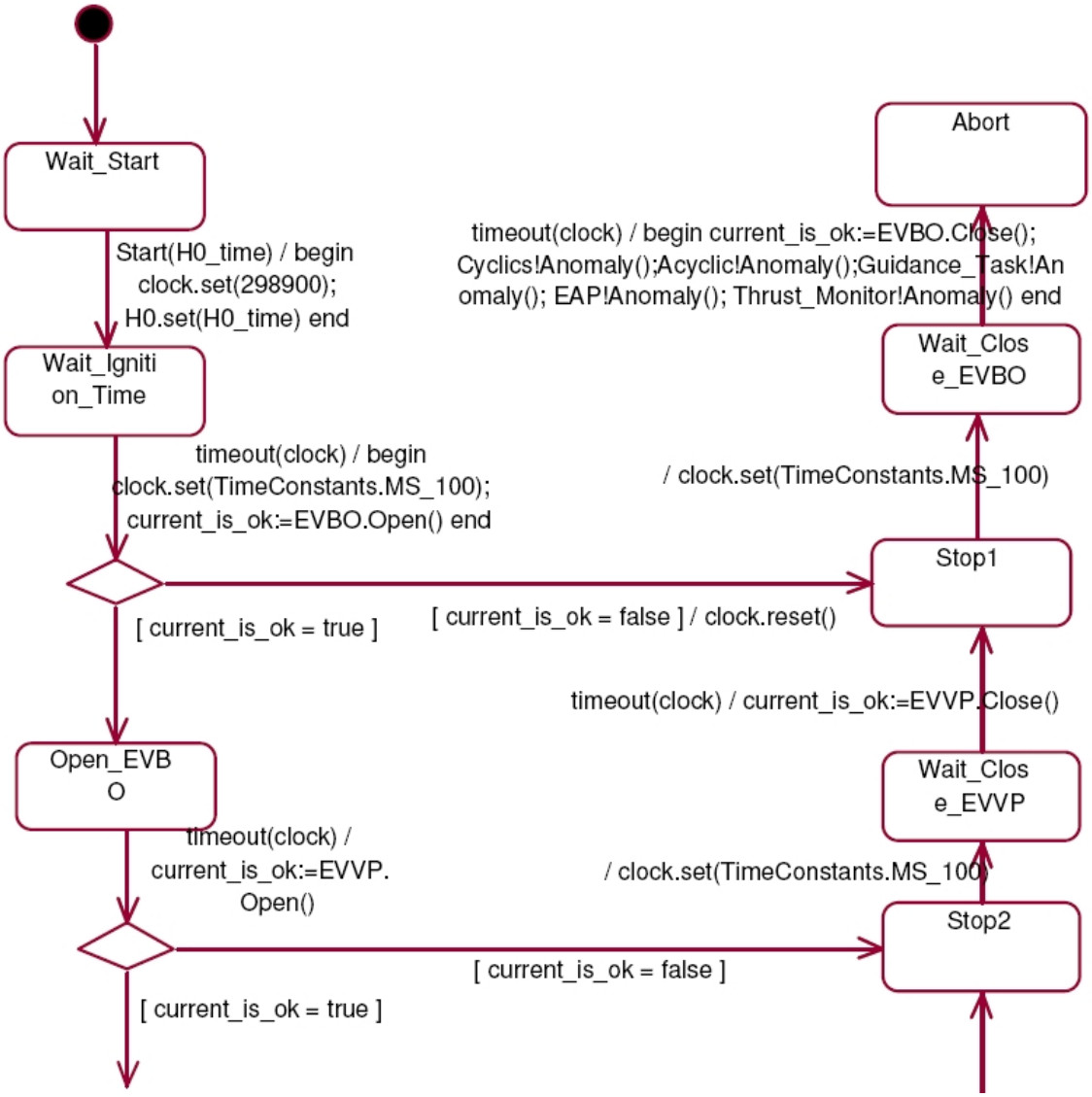
Component-based Design – Heterogeneity



Matlab/Simulink

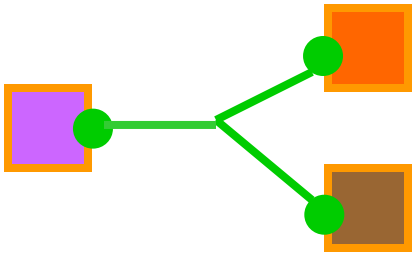
Component-based Design – Heterogeneity

UML Model
(Rational Rose)

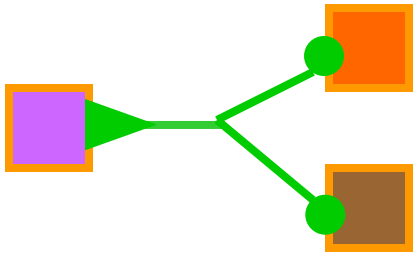


Component-based Design – Heterogeneity

Heterogeneity of Interaction



Rendezvous: atomic symmetric synchronization



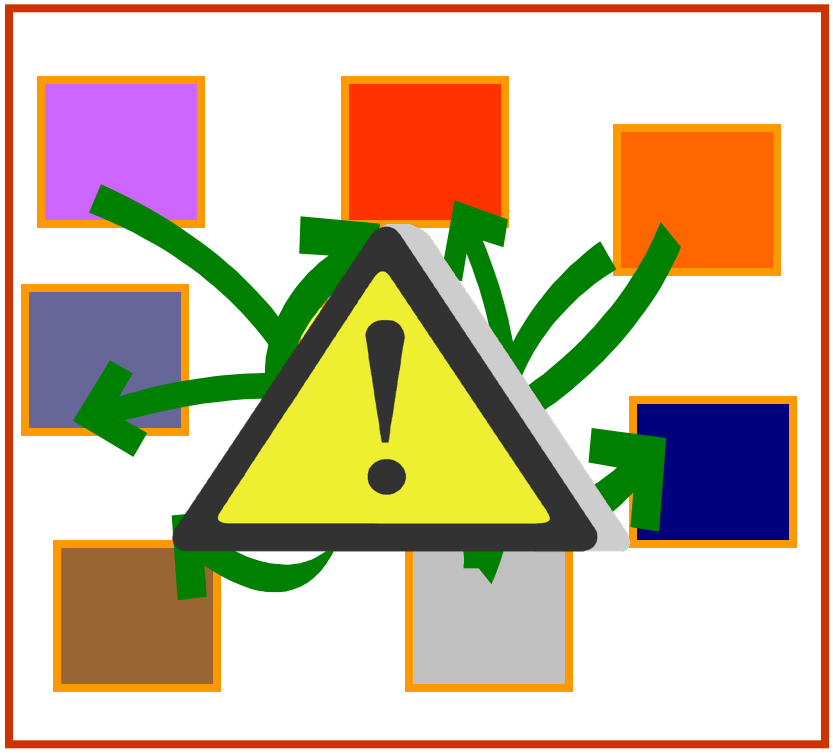
Broadcast: asymmetric synchronization triggered by a Sender

- Any interaction mechanism can be expressed as the hierarchically structured combination of rendezvous and broadcast
- Existing formalisms and theories are not expressive enough
  - they use variety of low-level coordination mechanisms including semaphores, monitors, message passing, function call
  - encompass point-to-point interaction rather than multiparty interaction

Component-based Design – Heterogeneity

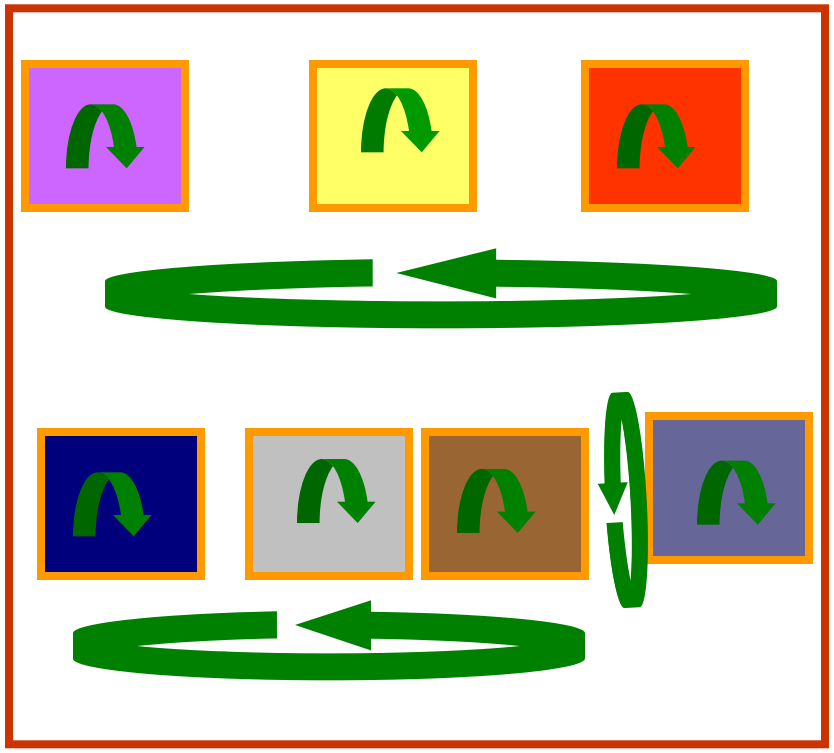
Heterogeneity of Programming Styles

Thread-based programming



Software Engineering

Actor-based programming

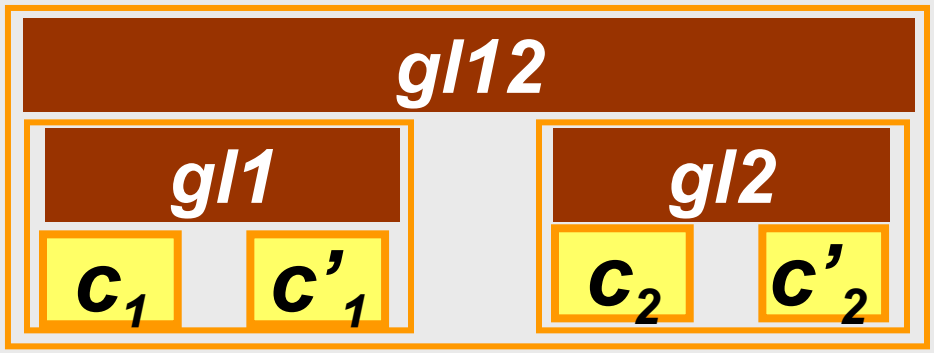


Systems Engineering

Component-based Design – The Concept of Glue

Build a component *C* satisfying a given property *P, from*

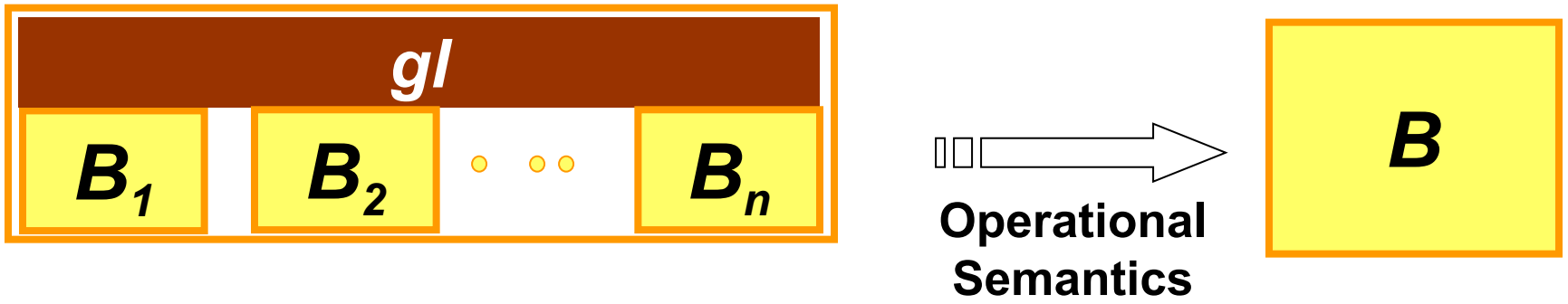
- $\mathcal{C}_0$ a set of **atomic** components described by their behavior
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of **glue *operators*** on components



- Glue operators are coordination mechanisms such as such as protocols, schedulers, buses
- We need a unified composition paradigm for describing and analyzing the coordination between components in terms of tangible, well-founded and organized concepts

Component-based Design – Glue Operators

We use operational semantics to define the meaning of a composite component – glue operators are “behavior transformers”



Glue Operators

- build interactions of composite components from the actions of the atomic components e.g. parallel composition operators
- can be specified by using a family of derivation rules (the Universal Glue)

Component-based Design – Glue Operators

A **glue operator** defines **interactions** as a set of derivation rules of the form

$$\frac{\{q_i - a_i \rightarrow_i q'_i\}_{i \in I} \qquad C(q_k)_{k \in K}}{(q_1, .., q_n) - a \rightarrow (q'_1, .., q'_n)}$$

- $I, K \subseteq \{1, \ldots n\}$, $I \neq \emptyset$, $K \cap I = \emptyset$
- $a = \bigcup_{i \in I} a_I$ is an interaction
- $q'_i = q_i$ for $i \notin I$

Notice that, non deterministic choice and sequential composition are not glue operators

A **glue** is a set of glue operators

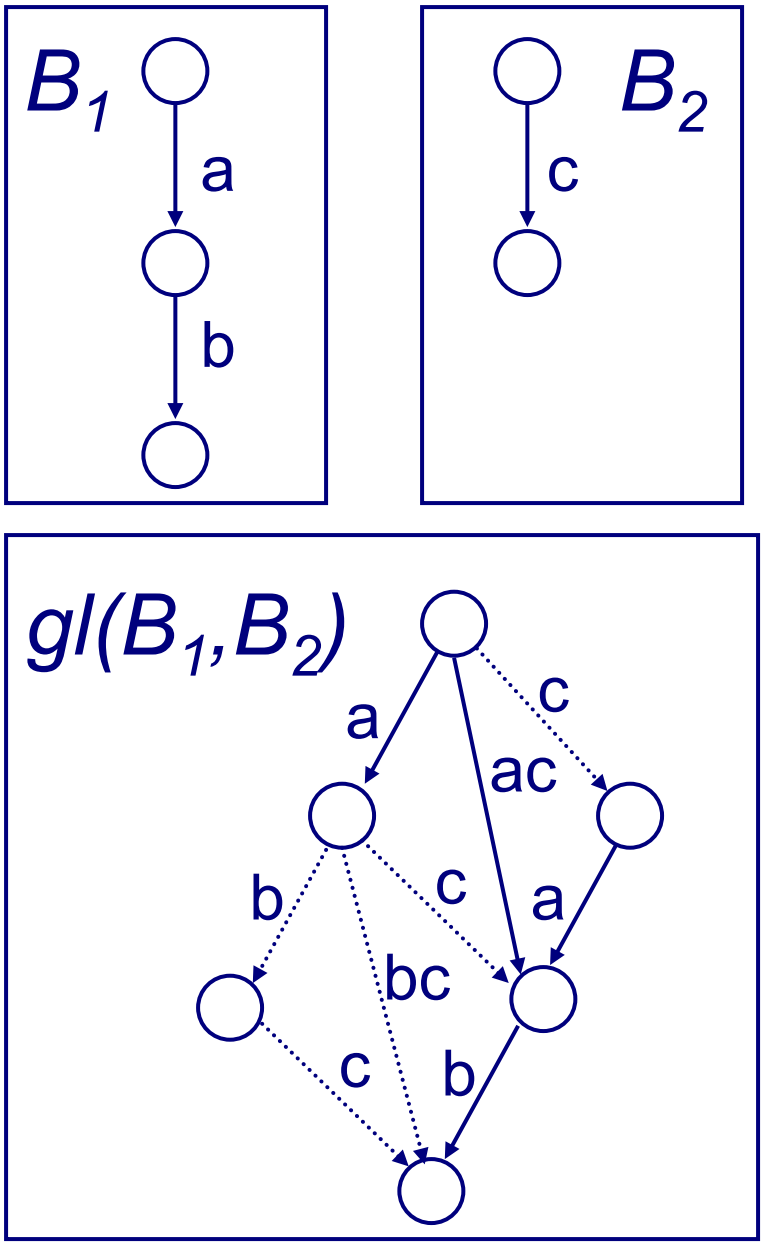
Component-based Design – Glue Operators: Example

gl is defined by

$$\frac{q_1 - a \rightarrow q'_1}{q_1 q_2 - a \rightarrow q'_1 q_2}$$

$$\frac{q_1 - a \rightarrow q'_1 \quad q_2 - c \rightarrow q'_2}{q_1 q_2 - ac \rightarrow q'_1 q'_2}$$

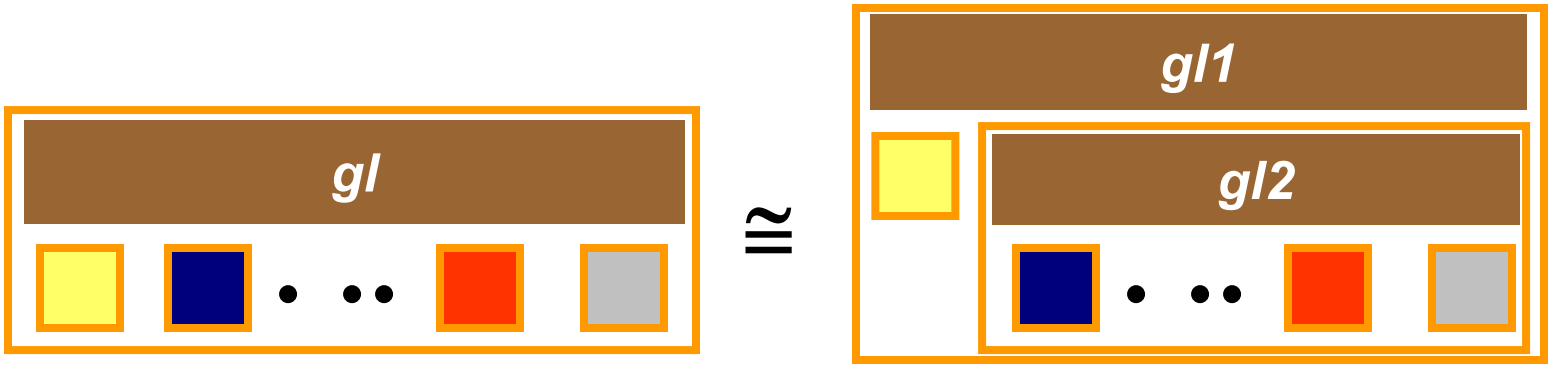
$$\frac{q_1 - b \rightarrow q'_1 \quad \neg\, q_2 - c \rightarrow}{q_1 q_2 - b \rightarrow q'_1 q_2}$$



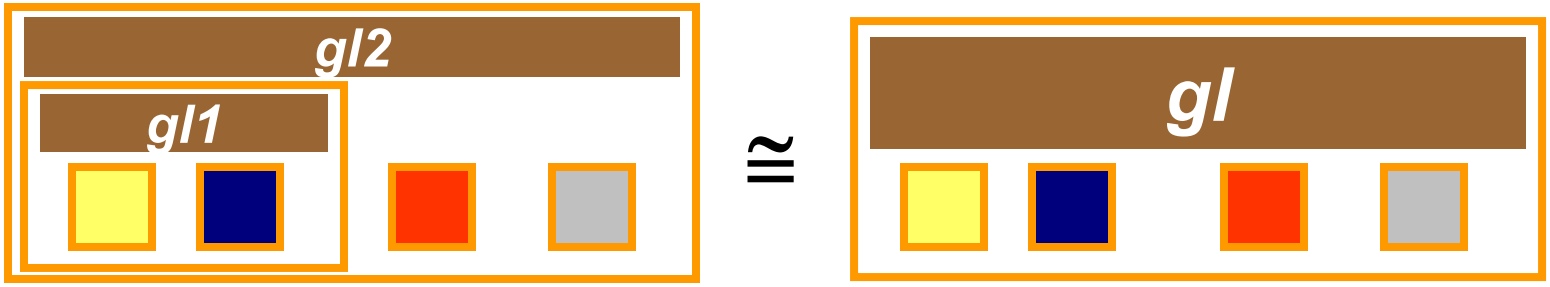
Component-based Design – Glue Operators: Properties

Glue is a first class entity independent from behavior that can be decomposed and composed

1. Incrementality



2. Flattening



Component-based Design – Glue Operators: Expressiveness

- Comparison between formalisms and models is done by flattening structure and reduction to behaviorally equivalent models e.g. finite state automaton, Turing machine

- This leads to notions of expressiveness that are **not adequate** for comparing coordination capabilities of languages and models e.g.
  - ❑ all finite state formalisms turn out to be expressively equivalent
  - ❑ all modeling and programming languages are Turing complete, while their coordination capabilities tremendously differ

Objective:

- Propose notions of expressiveness based on a strict separation between behavior and coordination
- Compare existing frameworks by using such notions

Component-based Design – Glue Operators: Expressiveness

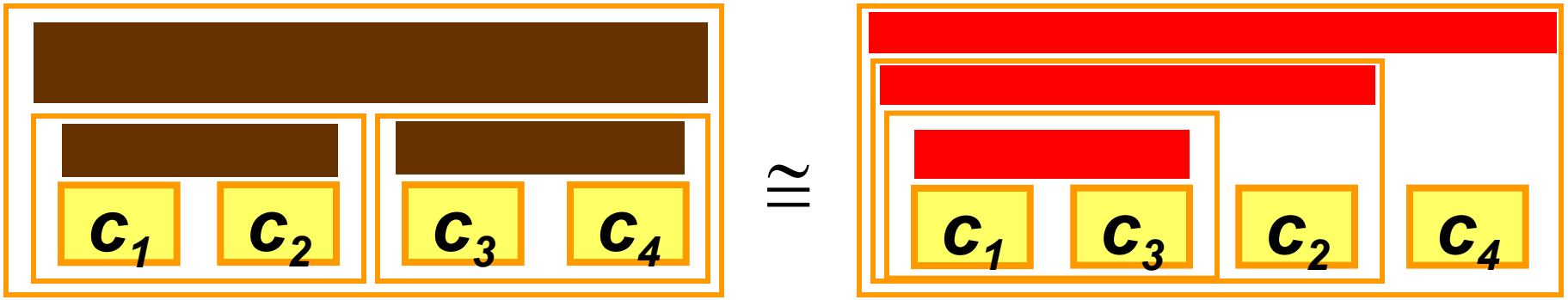
- Different from the usual notion of expressiveness!
- Based on strict separation between glue and behavior

Given two glues G_1 , G_2

G_2 ***is strongly more expressive than*** G_1

if for any component built by using G_1 and a set of components $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0$



Component-based Design – Glue Operators: Expressiveness

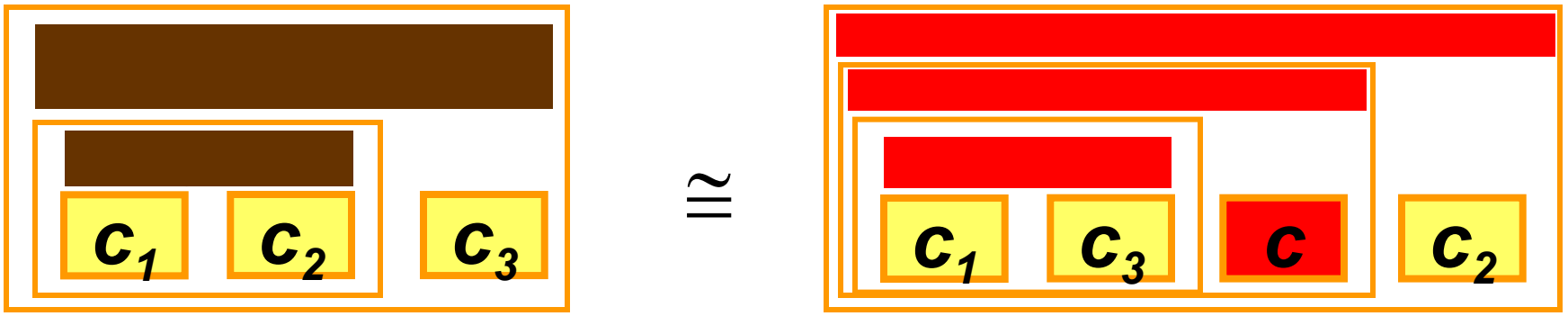
Given two glues G_1 , G_2

G_2 ***is weakly more expressive than*** G_1

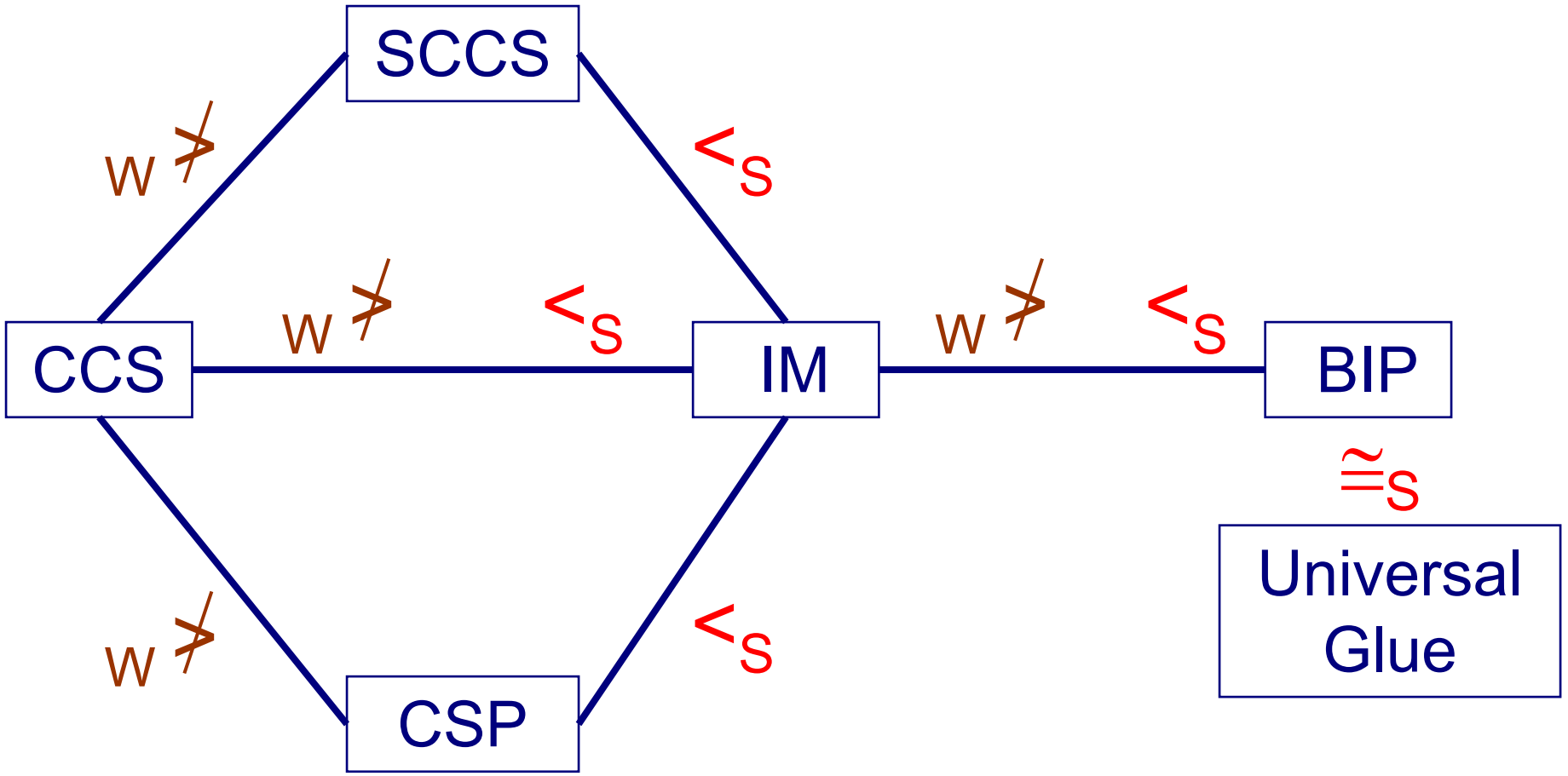
if for any component built by using G_1 and a set of components $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0 \cup \mathcal{C}$

where $\mathcal{C}$ is a finite set of coordinating components.



Component-based Design – Glue Operators: Expressiveness



[Bliudze&Sifakis, Concur 08]

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Rigorous System Design – Semantic Coherency

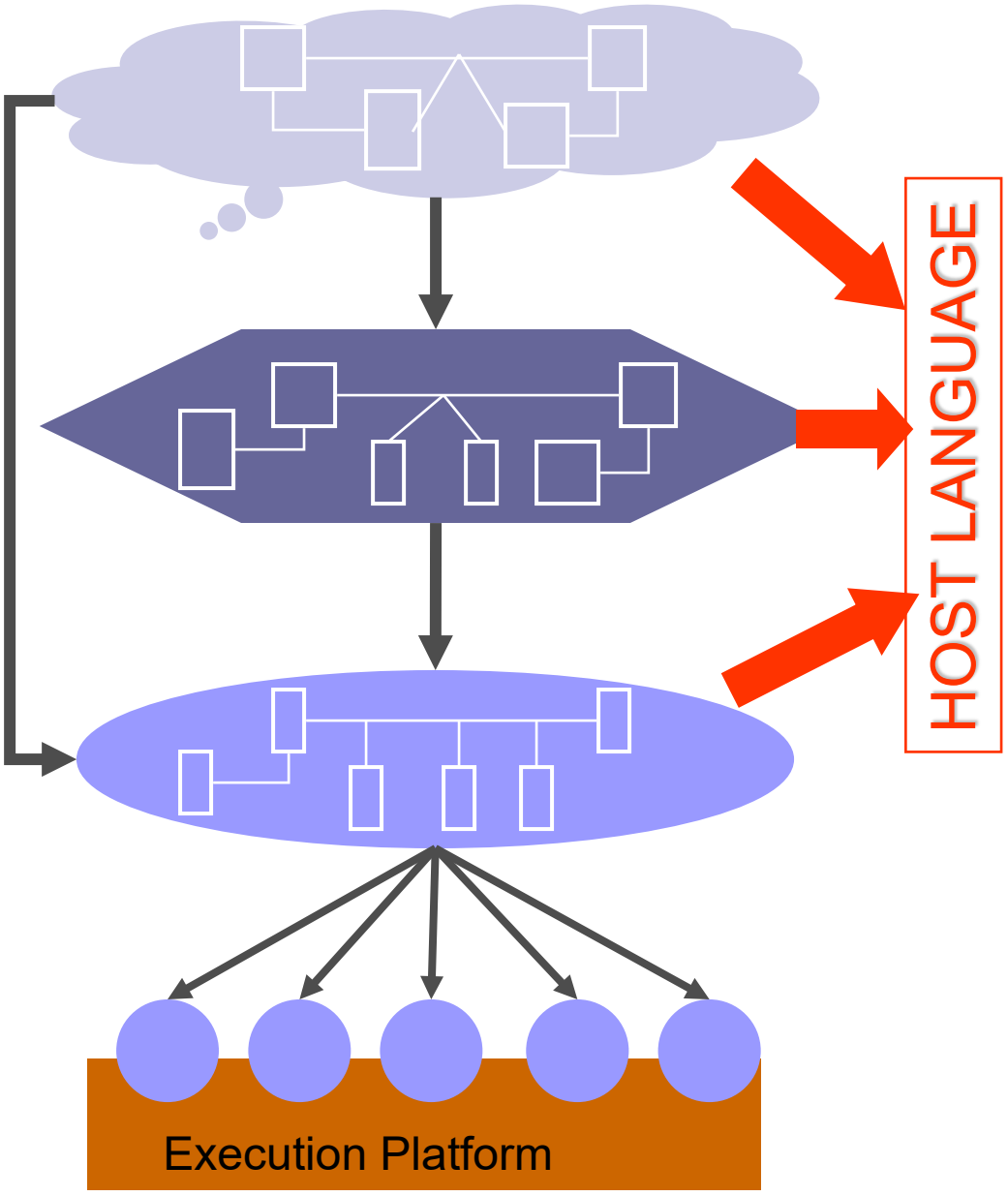
System designers deal with a large variety of languages, with different characteristics, each highlighting different dimensions of a system

TSpaces Fortran C Java OSGi
Softbench
SWbus
Corba
MPI
Javabeans
.NET
Fractal
Verilog VHDL SystemC TLM
BPEL
NesC
SES/Workbench
SysML
AADL
Statecharts
Matlab/Simulink

Consequences

- Using semantically unrelated formalisms e.g. for programming, HW description and simulation, breaks continuity of the design flow and jeopardizes its coherency
- System development is often decoupled from validation and evaluation.

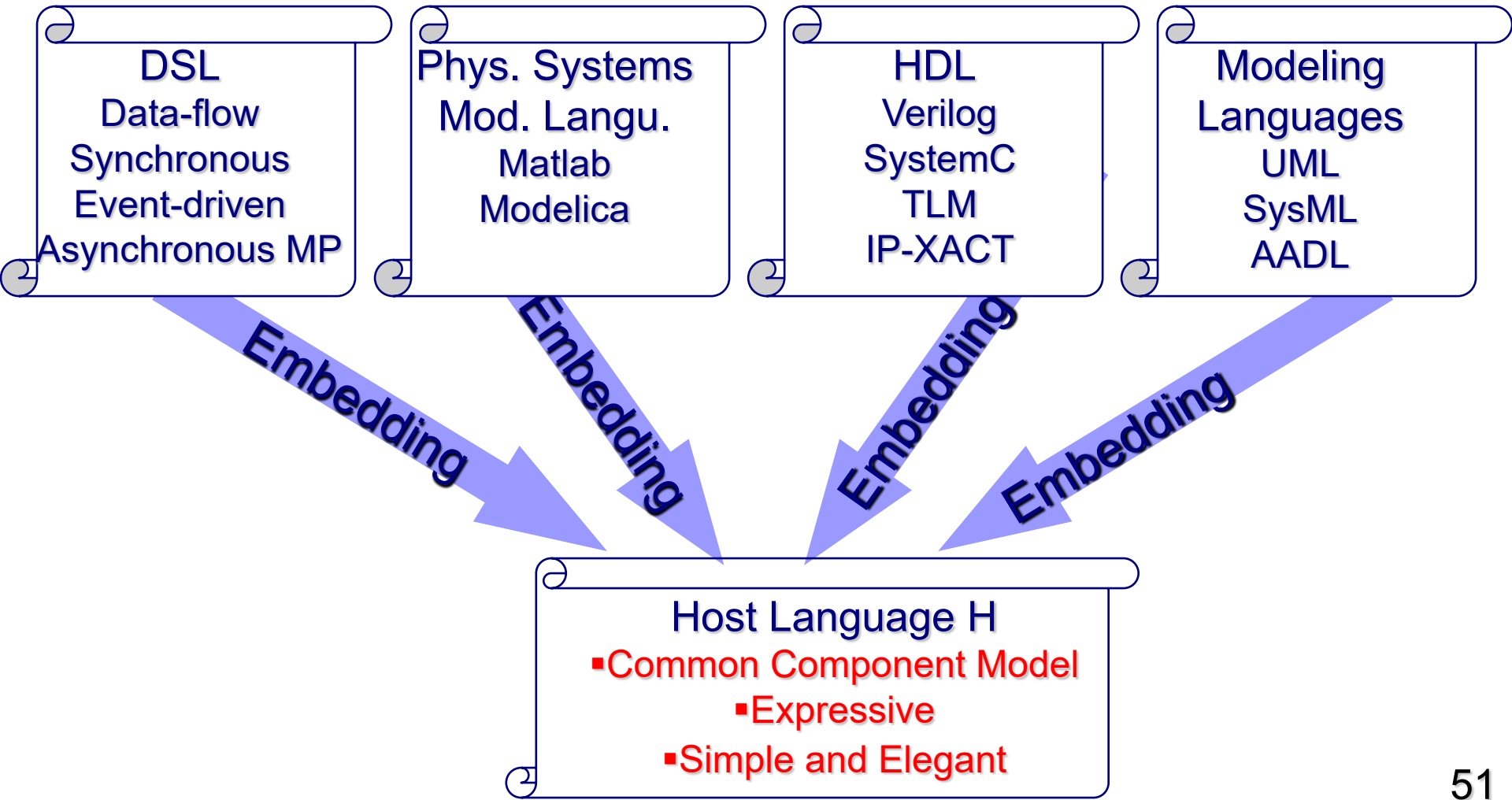
Semantic Coherency



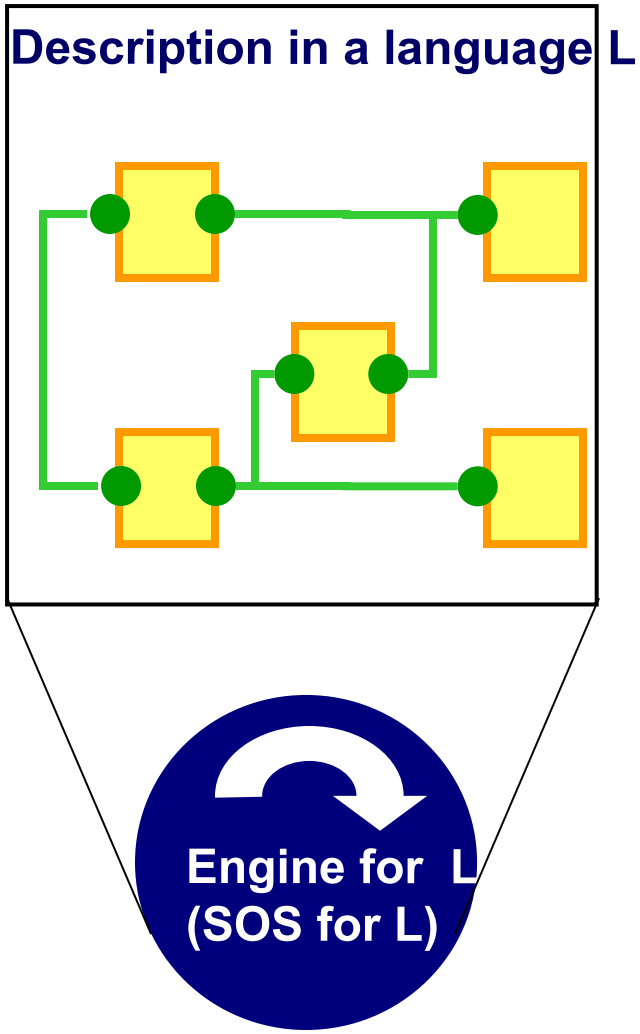
Any system design flow is de facto based on a host programming language such as C or Java

Semantic Coherency

To ensure global consistency of the design flow we need to express the semantics of the various languages in terms of an all encompassing host language



Rigorous System Design – Semantic Coherency

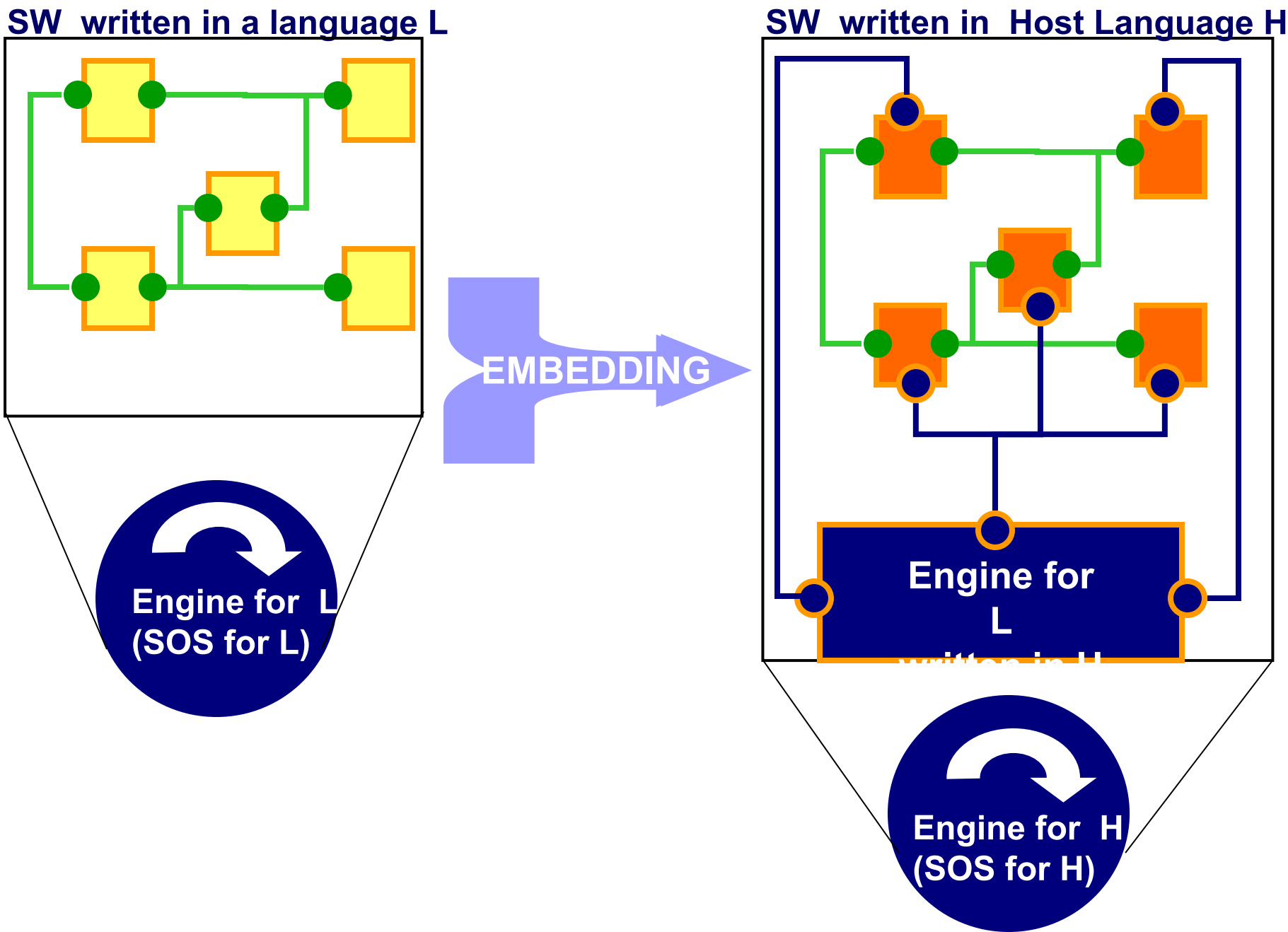


Structured Operational Semantics for L is implemented by an Engine which cyclically executes a two-phase protocol:

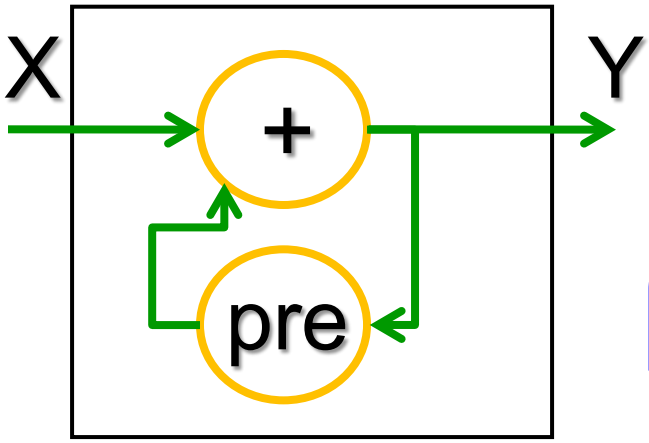
1. Monitors components and determines enabled interactions

2.Chooses and executes one enabled interaction

Rigorous System Design – Semantic Coherency

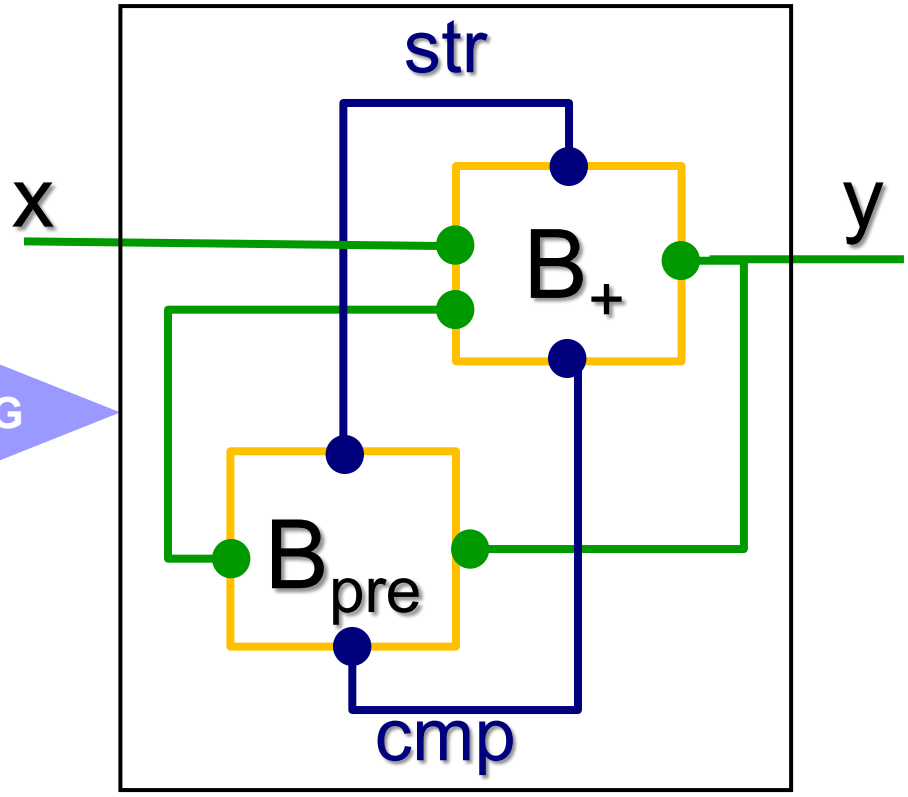


Rigorous System Design – Semantic Coherency



Y=X+pre(Y)
Program in Lustre

EMBEDDING



Program in BIP

Rigorous System Design – Semantic Coherency

Host Languages should be

- expressive – coordination glue between components e.g. protocols, schedulers, buses, architectures can be expressed as the combination of composition operators
- minimal simple and elegant - achievement of a given functionality with a minimum of mechanism and a maximum of clarity

Using Host Languages allows

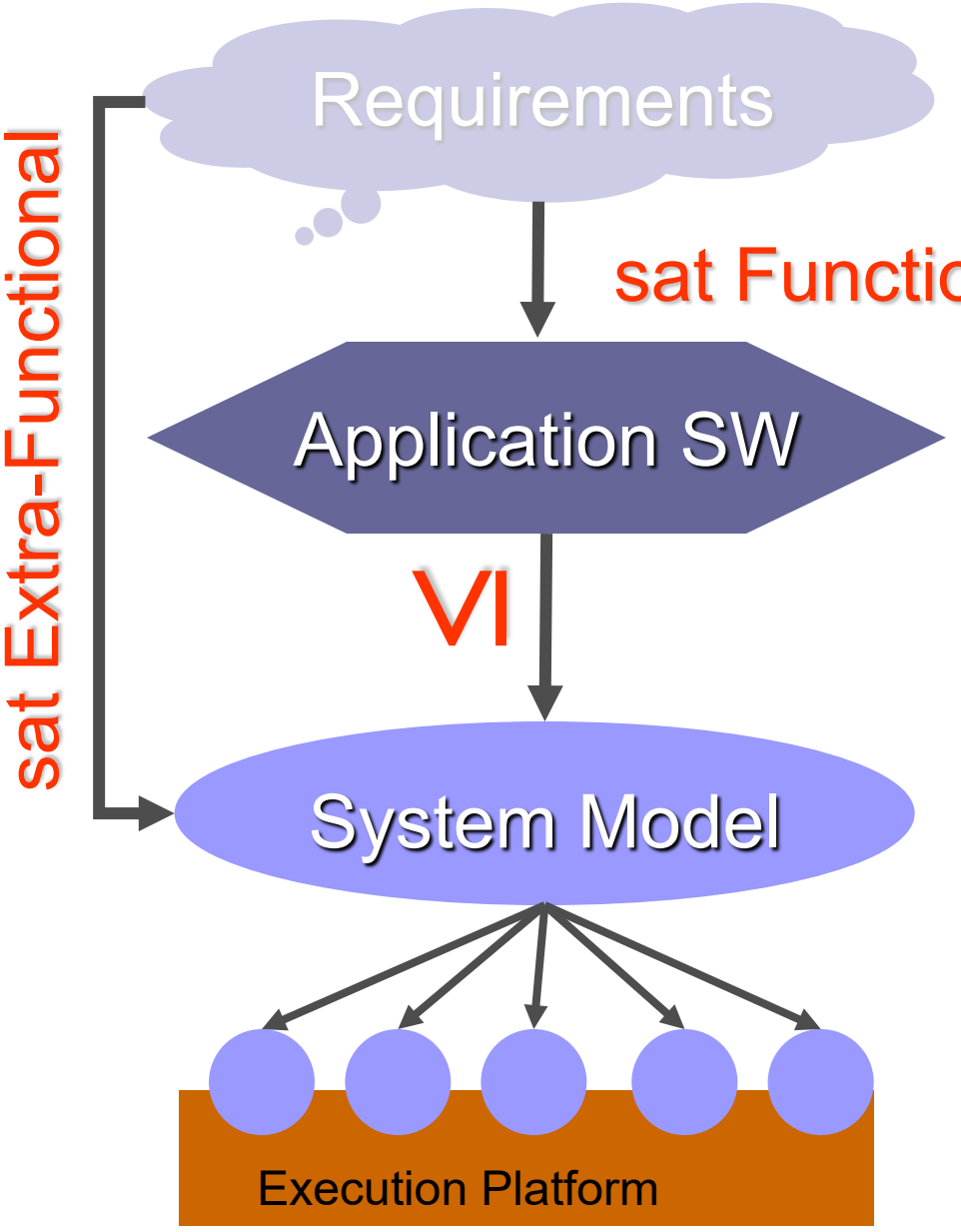
- overcoming the limitations of existing theoretical frameworks based on a single composition operator e.g. function call, asynchronous message passing, rendezvous

- unification of design flows through a Common Component Model

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

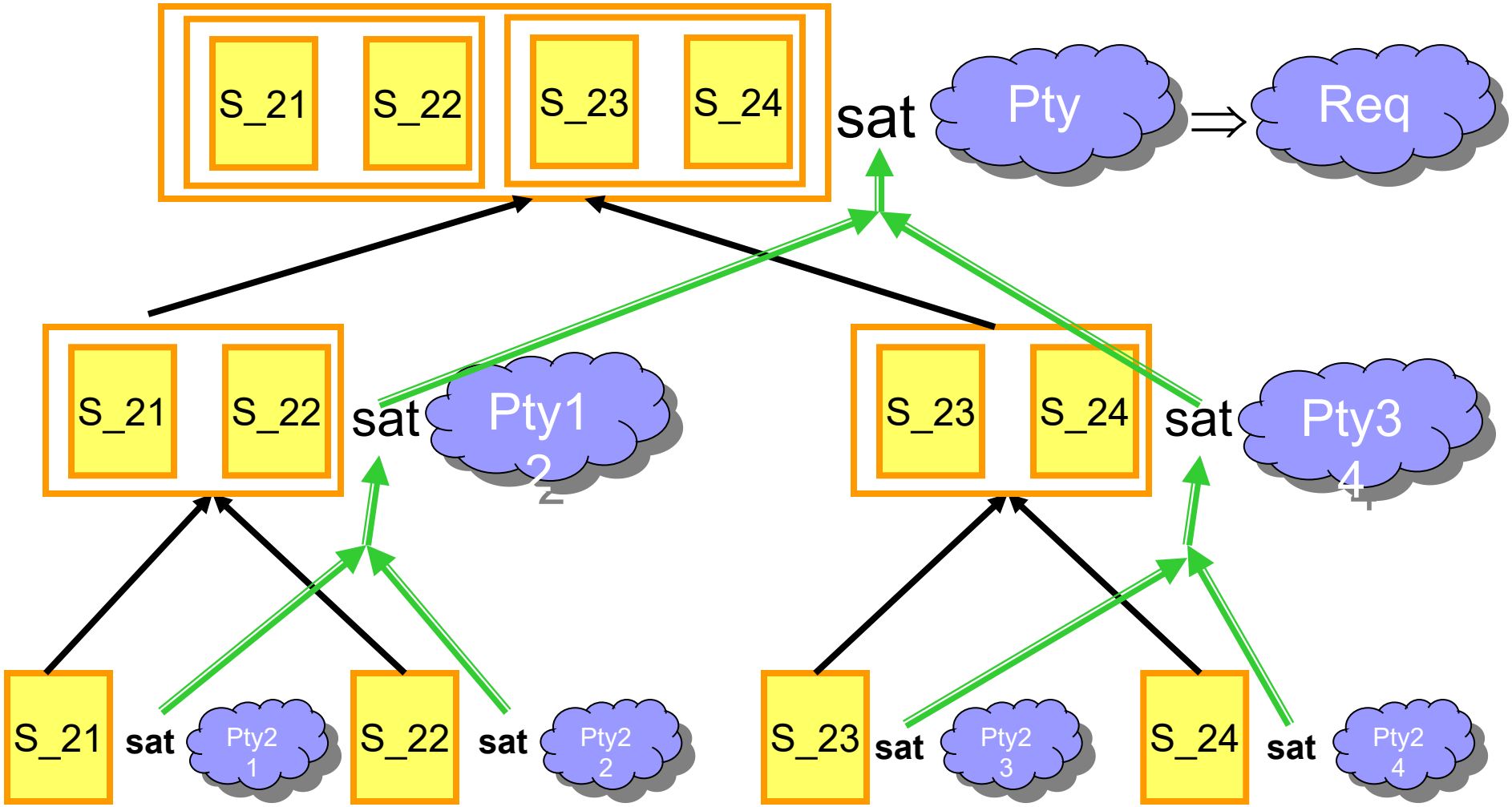
OVERVIEW

Correct-by-Construction



≥: refinement relation preserving functional properties

Correct-by-Construction – Building Correct Components

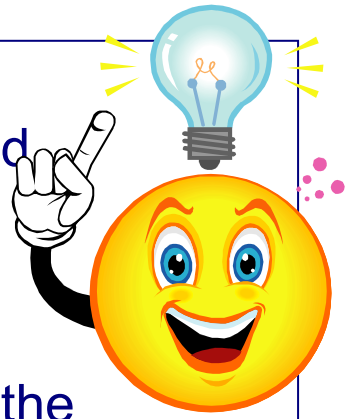


Base elements e.g. atomic components

Correct-by-Construction – Architectures

Architectures

- depict design principles, paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes



- are a means for ensuring global properties characterizing the coordination between components – correctness for free

- Using architectures is key to ensuring trustworthiness and optimization in networks, OS, middleware, HW devices etc.

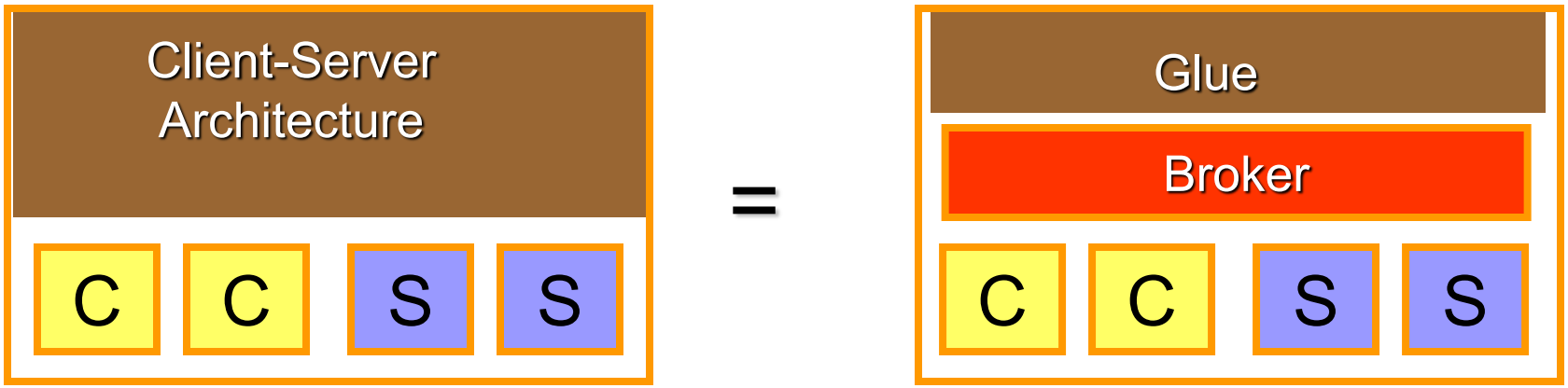
System developers extensively use libraries of reference architectures ensuring both functional and non functional properties e.g.

- Fault-tolerant architectures
- Resource management and QoS control
- Time-triggered architectures
- Security architectures
- Adaptive Architectures

Correct-by-Construction – Architectures

An architecture is a component transformer A(n)[X] and a characteristic property P(n), parameterized by an integer n such that

- A(n)[C1,..,Cn] = gl(n) (C1,..,Cn, D(n)), where D(n) is a set of coordinating components
- A(n)[C1,..,Cn] meets the characteristic property P(n).

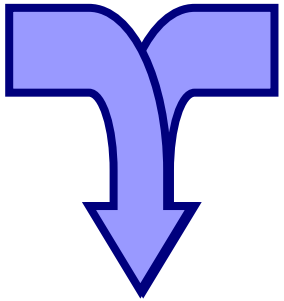


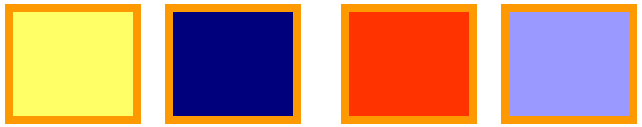
Characteristic property: atomicity of transactions, fault-tolerance ….

Correct-by-Construction – Architectures

Rule1: Property Preservation

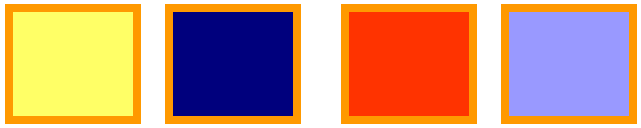
Deadlock-free
Routing Protocol





Deadlock-free components

Deadlock-free
Routing Protocol

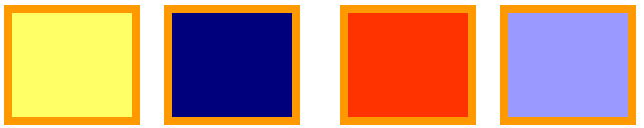


Deadlock-free
composite
component

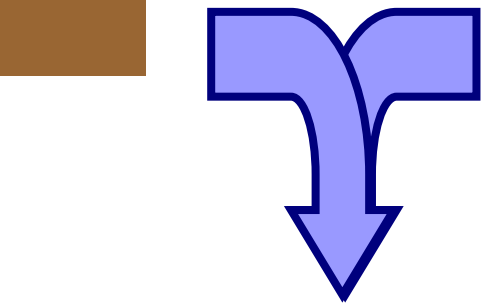
Correct-by-Construction – Architectures

Rule2: Property Enforcement

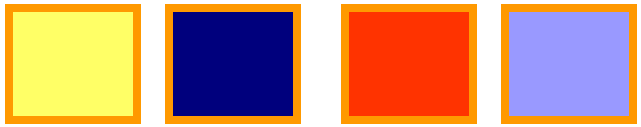
Architecture
for Mutual Exclusion



Components



Architecture
for Mutual Exclusion



satisfies Mutex

Correct-by-Construction – Architectures: Composability

An architecture ensuring a given property can be obtained as the combination of a set of architectures ensuring basic properties.

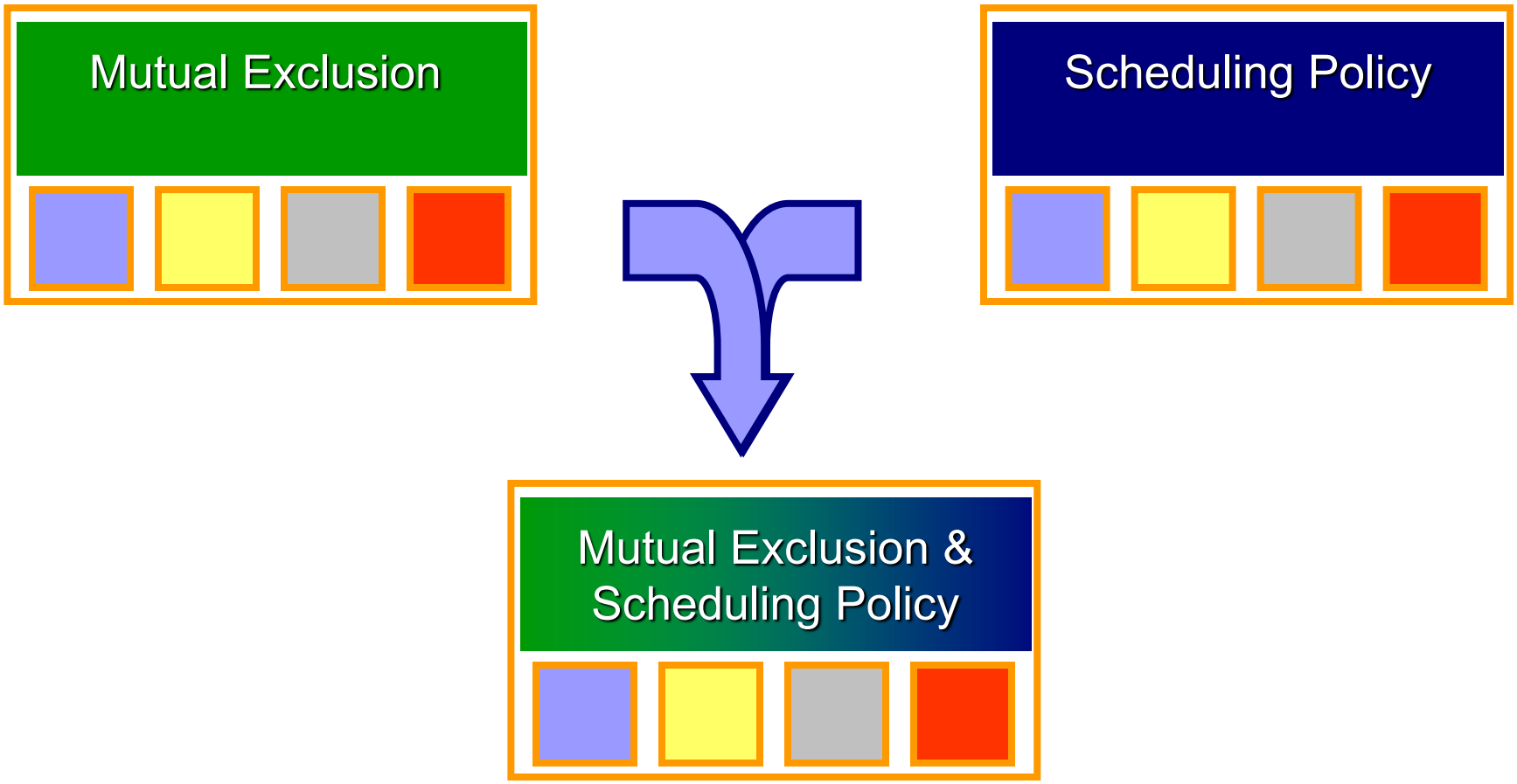
For example, security architectures are obtained by composition of architectures ensuring

- Antivirus protection
- Intrusion Detection System, Intrusion Protection System
- Sampling
- Monitoring
- Watermarking
- Embedded cryptography
- Integrity checking

Composability: We need theory for combining basic architectures and their characteristic properties to obtain an architecture meeting a given global property

Correct-by-Construction – Architectures: Composability

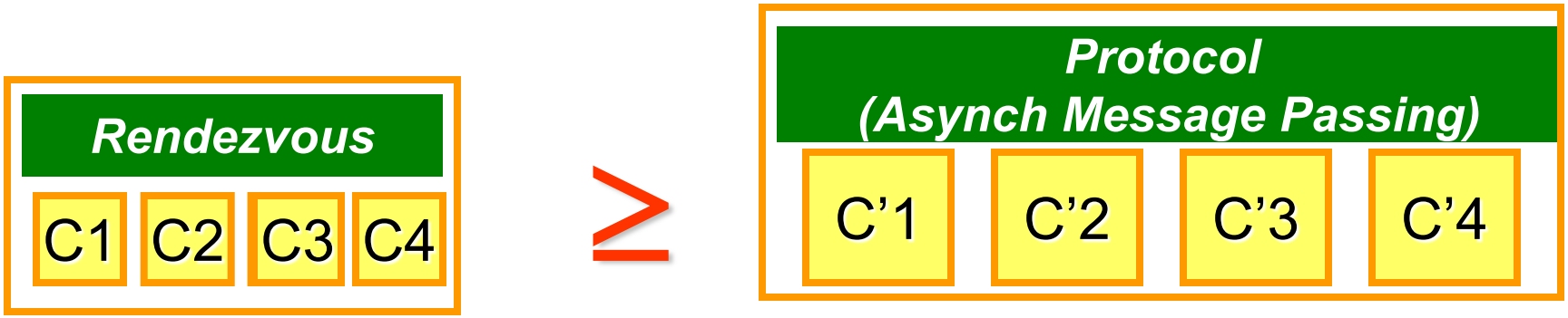
Rule3: Property Composability



Feature interaction in telecommunication systems, interference among web services and interference in aspect programming are all manifestations of a lack of composability

Correct-by-Construction: Refinement

The Refinement Relation ≥



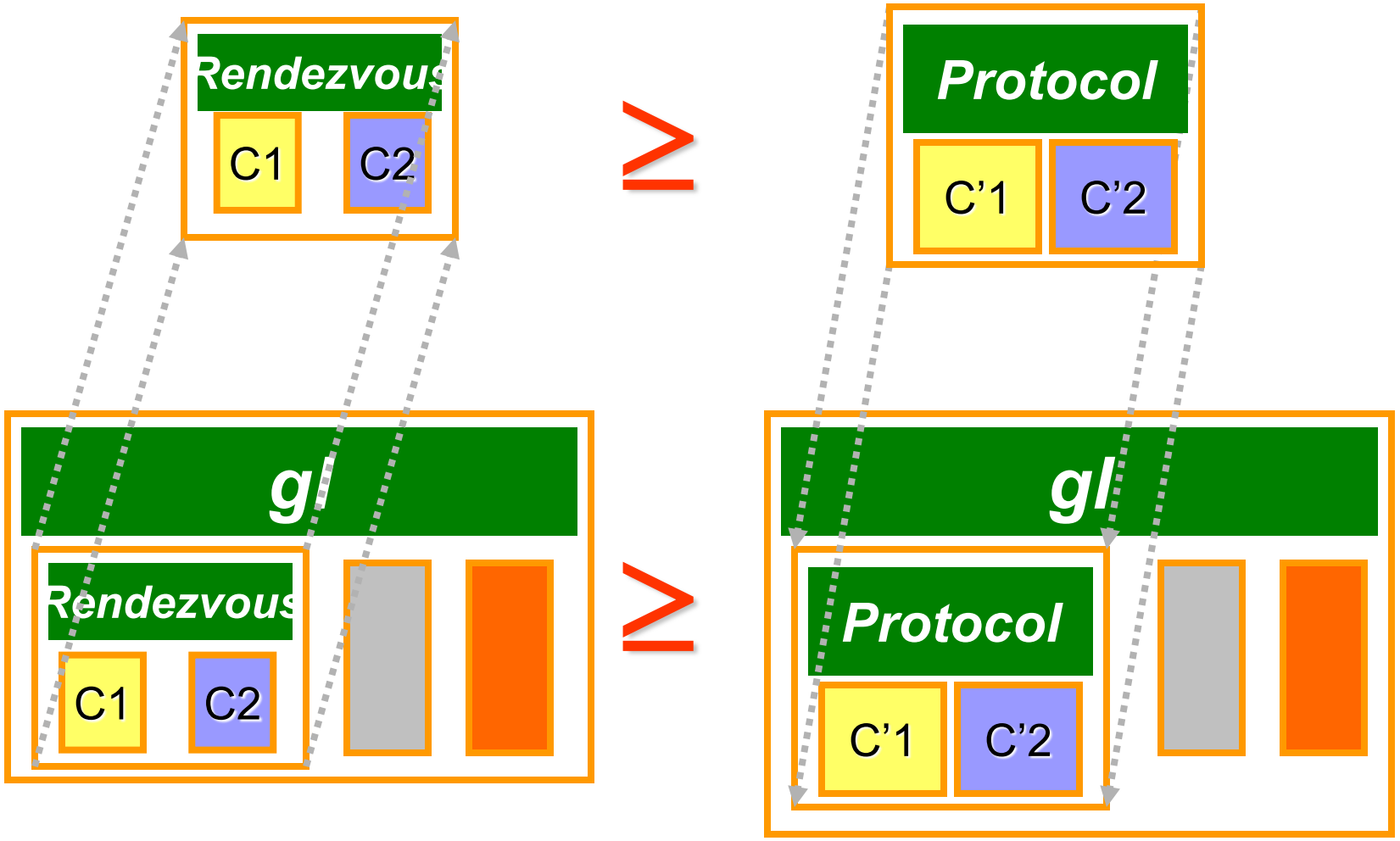
S1 S2

S1 ≥ S2 (S2 refines S1) if

- all traces of S2 are traces of S1 (modulo some observation criterion)
- if S1 is deadlock-free then S2 is deadlock-free too
- ≥ is preserved by substitution

Rigorous System Design – Correctness-by-Construction for ≥

Preservation of ≥ by substitution



Rigorous System Design – Correctness-by-Construction for ≥

From the Application Software to the System model

The AS is written in high level languages
supporting abstractions such as

- Atomicity of primitives and interactions between components – in particular multiparty interaction
- A logical notion of time assuming zero-time actions and synchrony of execution wrt to the physical environment

IV

Source-to-source transformation in the host language

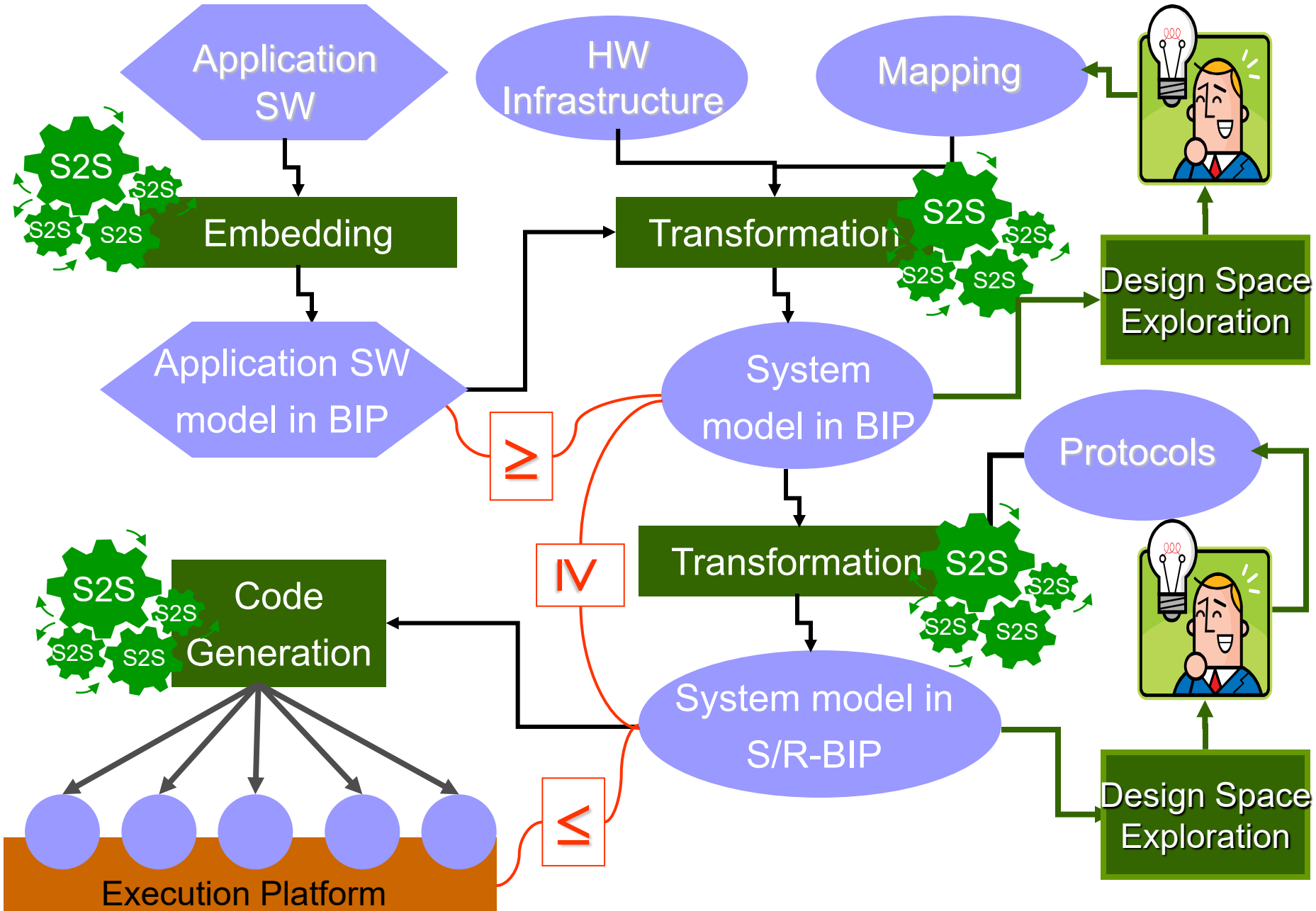
The generated system model
is a refinement of the AS generated automatically
for a given mapping associating

- Processes of the ASW → processors of the platform
- Data of the ASW → memories of the platform
- Interactions → execution paths or protocols

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

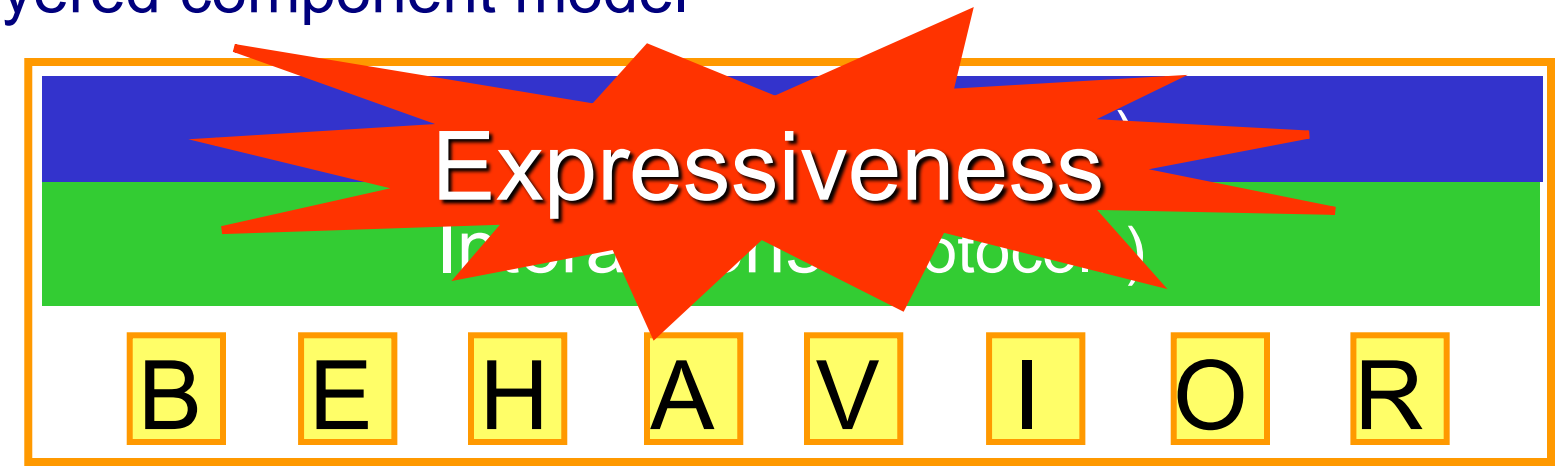
OVERVIEW

Putting RiSD into Practice – System Design in BIP

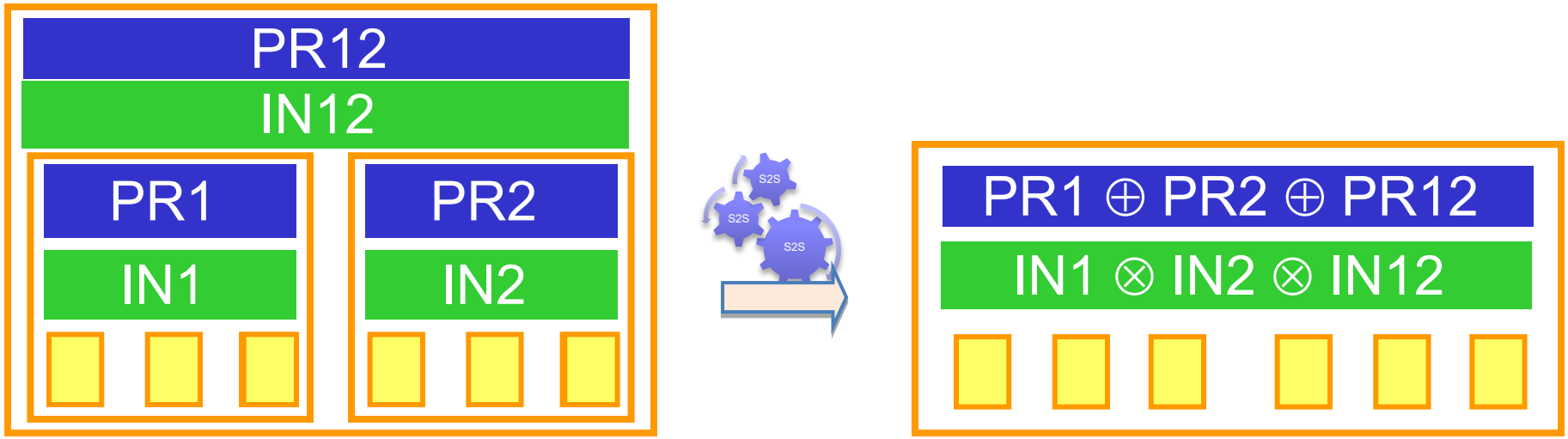


Modeling in BIP – Basic Concepts

Layered component model



Composition operation parameterized by glue IN12, PR12



Modeling in BIP – Semantics

- a set of atomic components $\{B_i\}_{i=1..n}$ where $B_i = (Q_i, 2^{Pi}, \rightarrow_i)$
- a set of interactions $\gamma \in 2^{2P}$ with $P = \cup_{i=1..n} P_i$ and $P_i \cap P_j = \emptyset$ $P = \cup_{i=1..n} P_i$
- a strict partial order $\pi \subseteq 2^P \times 2^P$

$\pi\, \gamma\, (B1,.,\, Bn)$

Interactions

$$\frac{\{a_i\}_{i\in I} \in \gamma \quad \{q_i - a_i \rightarrow q'_i\}_{i\in I} \quad a = \cup_{i\in I} a_i}{(q_1,.,\, q_n) - a \rightarrow_\gamma (q'_1,.,\, q'_n) \text{ where } q'_I = q_I}$$

Priorities

$$\frac{q - a \rightarrow_\gamma q' \quad \neg (\exists\, q - b \rightarrow_\gamma \wedge\, a\, \pi\, b)}{q - a \rightarrow_\pi q'}$$

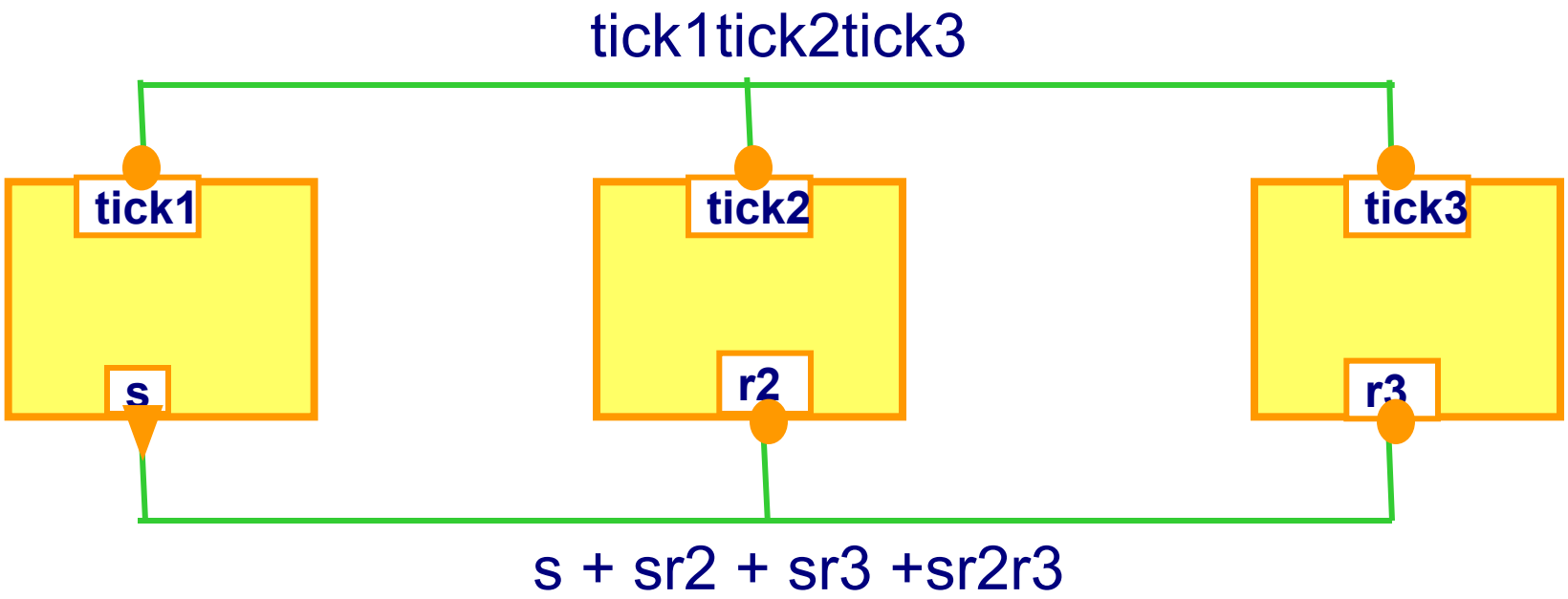
- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Modeling in BIP – Connectors

Express interactions by combining two protocols: rendezvous and broadcast

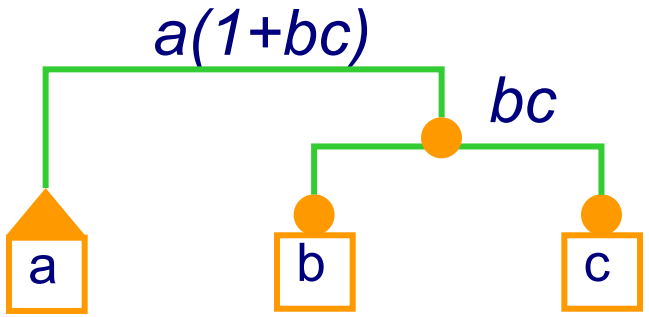
- A ***connector*** is a set of ports that can be involved in an interaction
- Port attributes (***trigger*** ▼, ***synchron*** ●) are used to model rendezvous and broadcast.
- An ***interaction*** of a connector is a set of ports such that: either it contains some trigger or it is maximal.



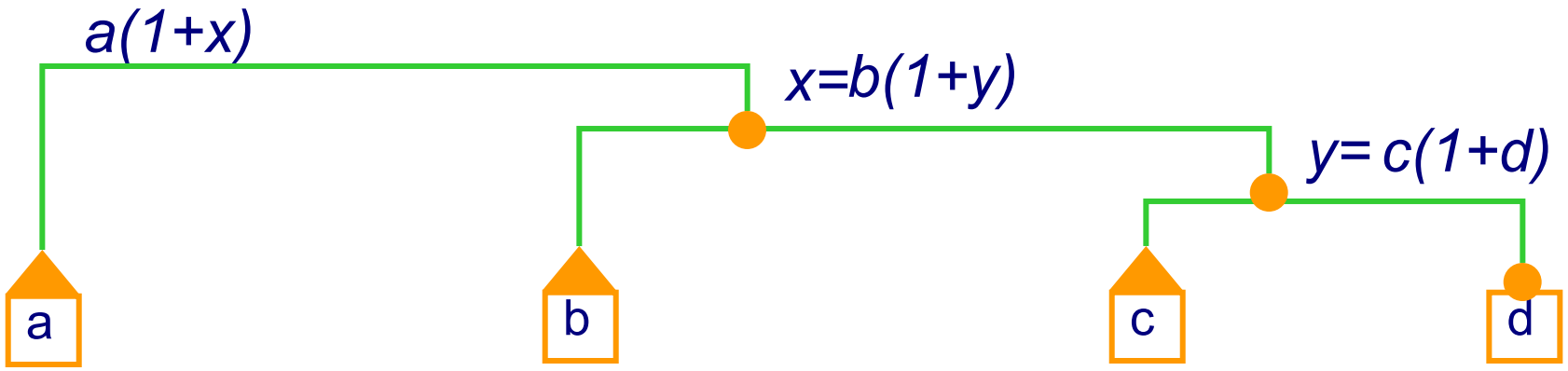
Modeling in BIP – Connectors

Atomic Broadcast:

a+abc



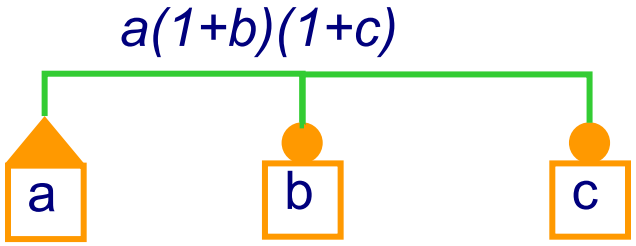
Causality chain: a+ab+abc+abcd



Modeling in BIP – Connectors

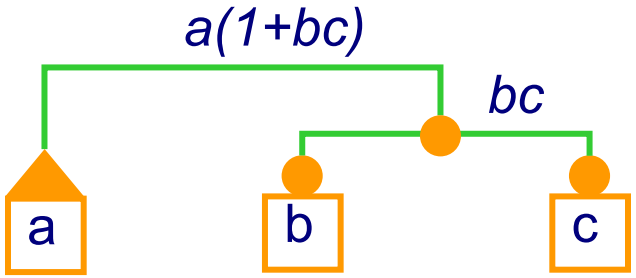
Broadcast

a’bc



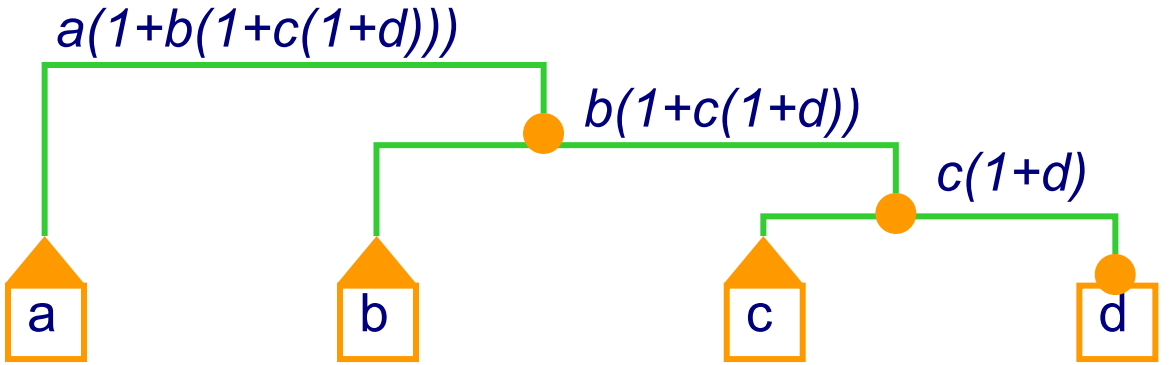
Atomic Broadcast

a’[bc]

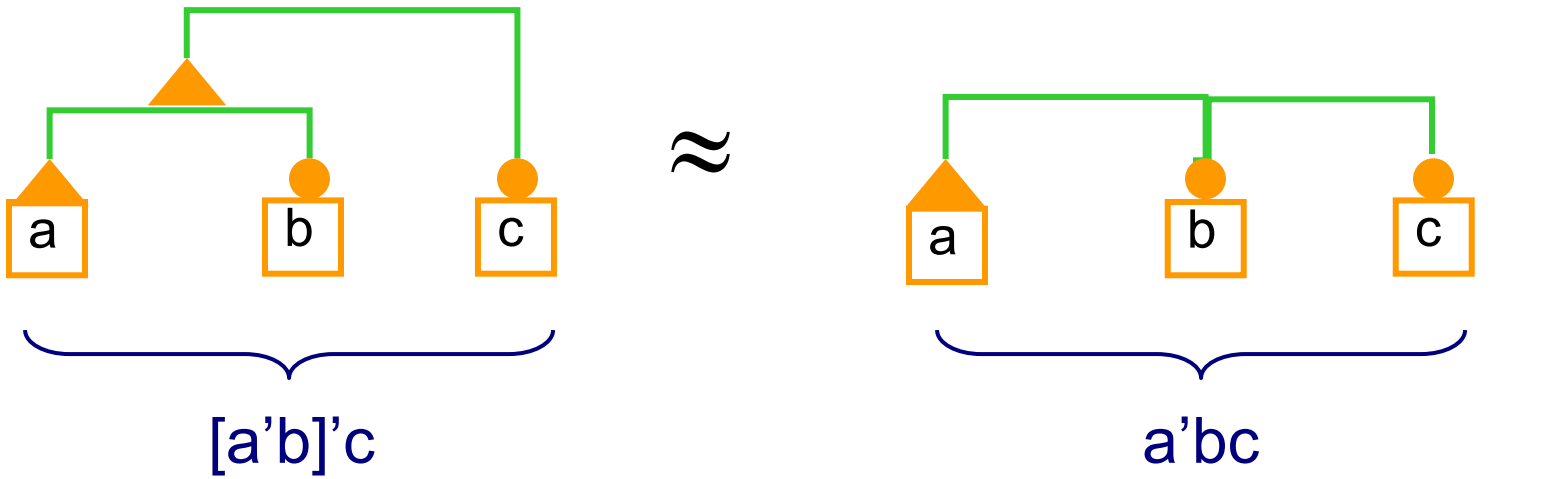


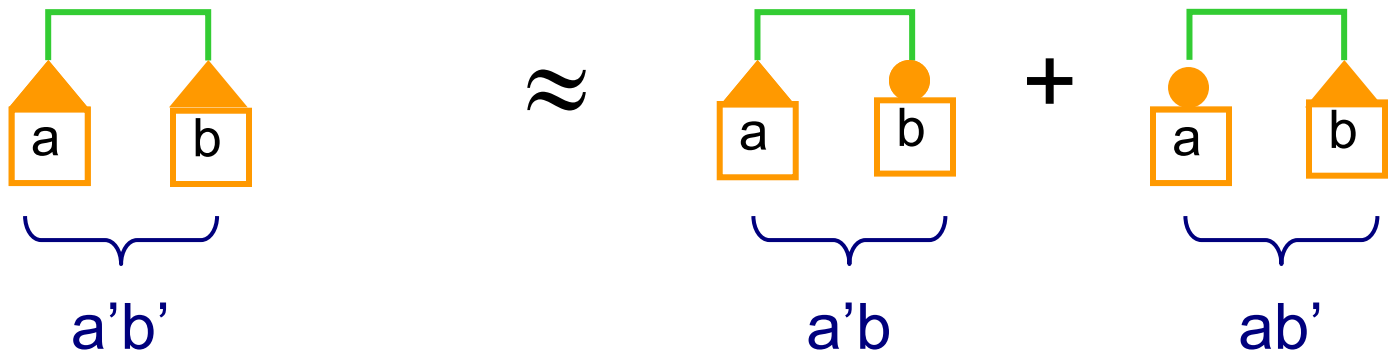
Causality chain

a’[b’[c’d]]



Modeling in BIP – Connectors





Modeling in BIP –The Algebra of Connectors

Syntax: s ::= [0] | [1] | [p] |[x] (synchrons)

t ::= [0]' | [1]' | [p]' | [x]' (triggers)

x ::= s | t | x.x | x + x

where P is a set of ports, such that $0,1 \notin P$

+	*union*	idempotent, associative, commutative, identity [0]
.	*fusion*	idempotent, associative, commutative, identity [1], distributive wrt + ([0] is not absorbing)
[ ], [ ]'	*typing*	unary operators

Semantics: defined as a function | |: AC(P) $\rightarrow$ 2^{2P}

Results *[Bliudze&Sifakis, EmSoft 07]*:

- Axiomatization
- Boolean representation allowing efficient implementation

The Algebra of Connectors – Boolean Representation

β: $AC(P) \rightarrow B(P)$ where $B(P)$ the boolean calculus on P

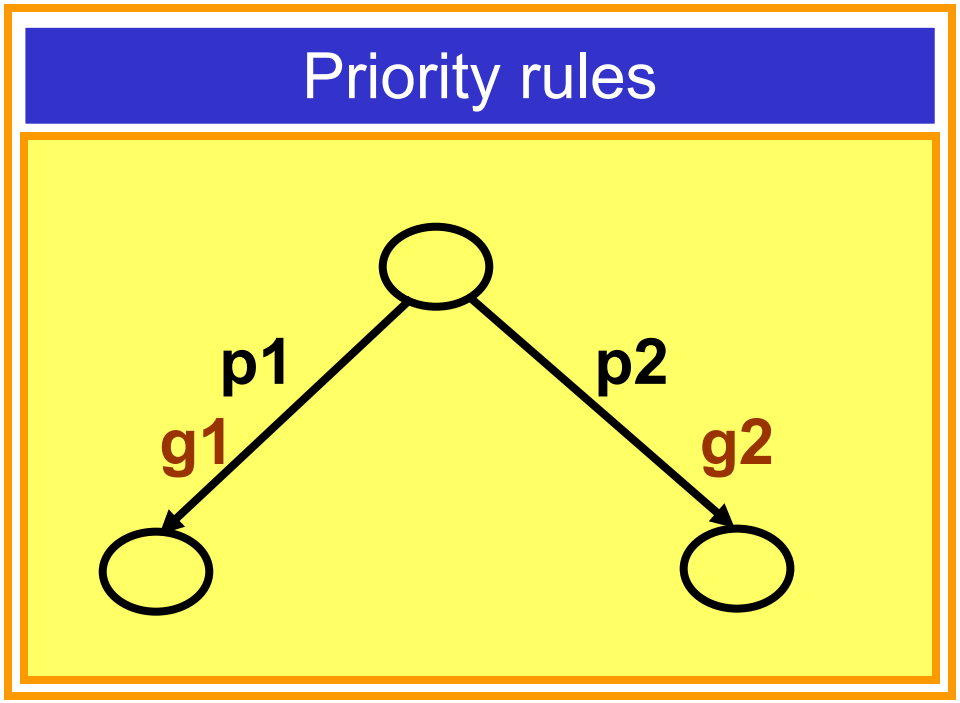
For $P=\{p,q,r,s,t\}$

$\beta(pq)$	$= p \wedge q \wedge \neg r \wedge \neg s \wedge \neg t$
$\beta(p'qr)$	$= p \wedge \neg s \wedge \neg t$
$\beta(p+q)$	$=(p \wedge \neg q \vee \neg p \wedge q) \wedge \neg r \wedge \neg s \wedge \neg t$
$\beta(0)$	= false
$\beta(1)$	$= \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t$
$\beta(1+p'q'r's't')$	= true

Results:

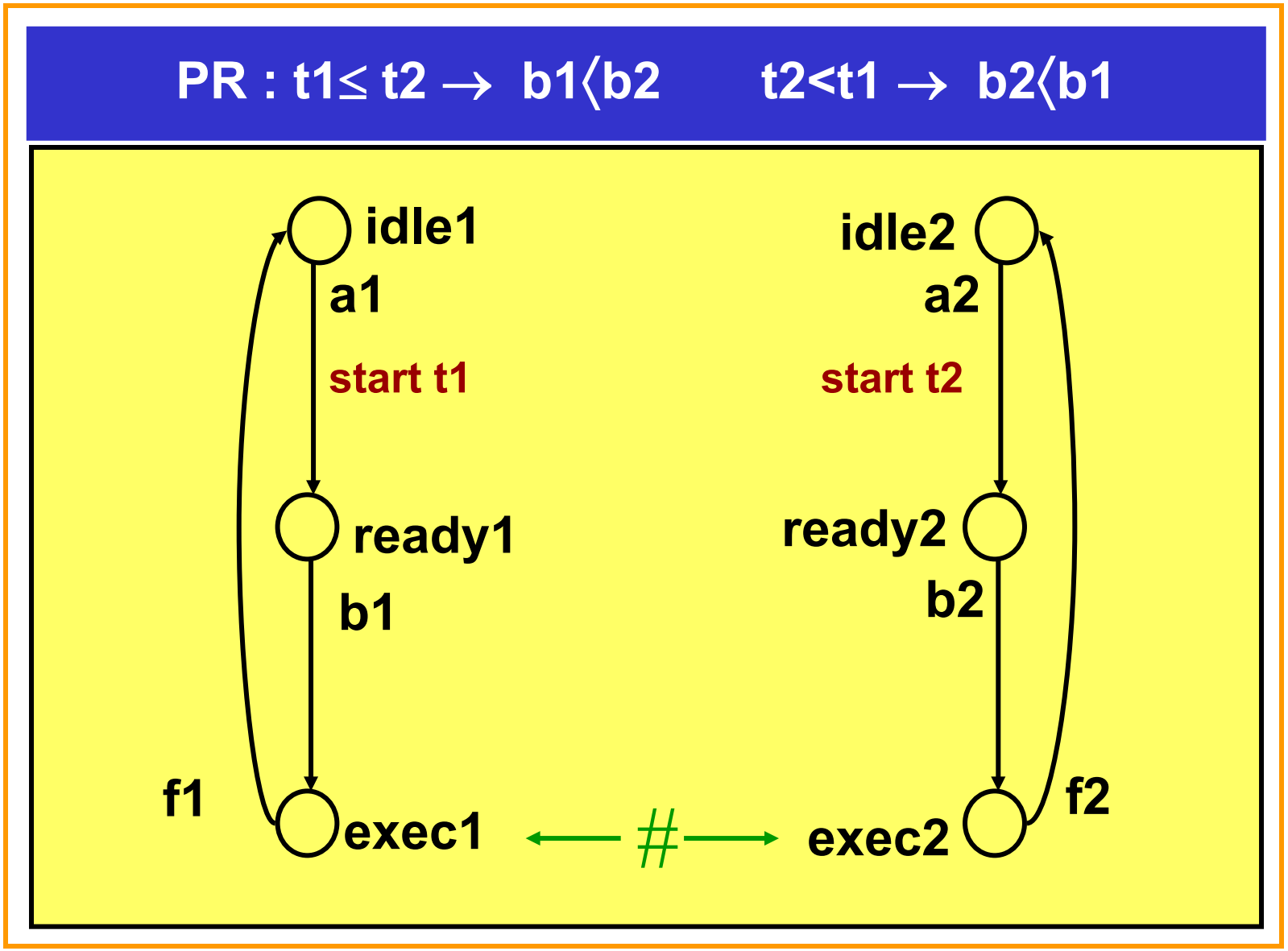
- *Efficient implementation of connectors by using BDDs*
- *Synthesis of connectors from boolean constraints on ports*

Modeling in BIP – Priorities



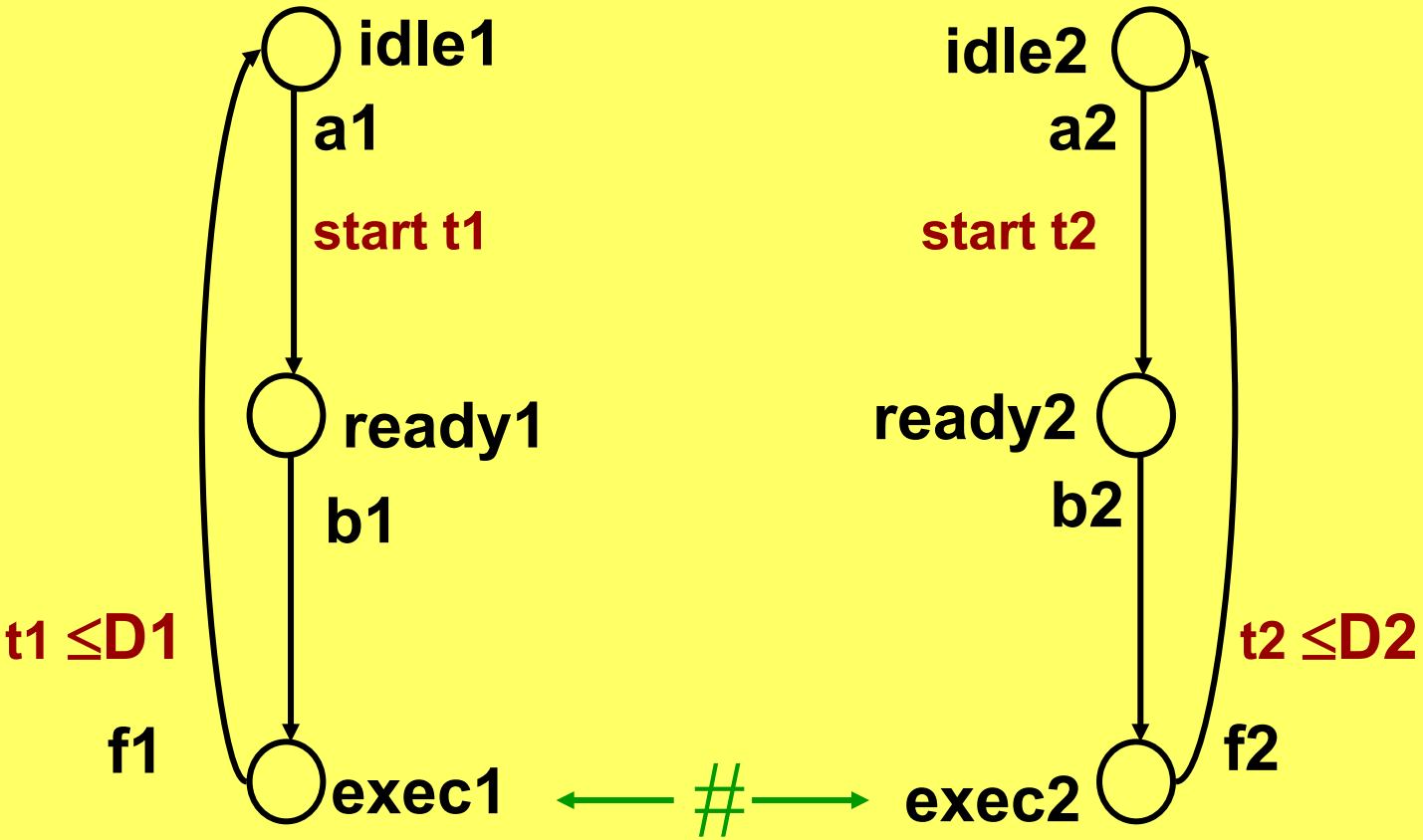
Priority rule	Restricted guard g1’
true $\rightarrow$ p1 $\langle$ p2	g1’ = g1 $\wedge \neg$ g2
C $\rightarrow$ p1 $\langle$ p2	g1’ = g1 $\wedge \neg$(C $\wedge$ g2)

Modeling in BIP – Priorities: FIFO policy

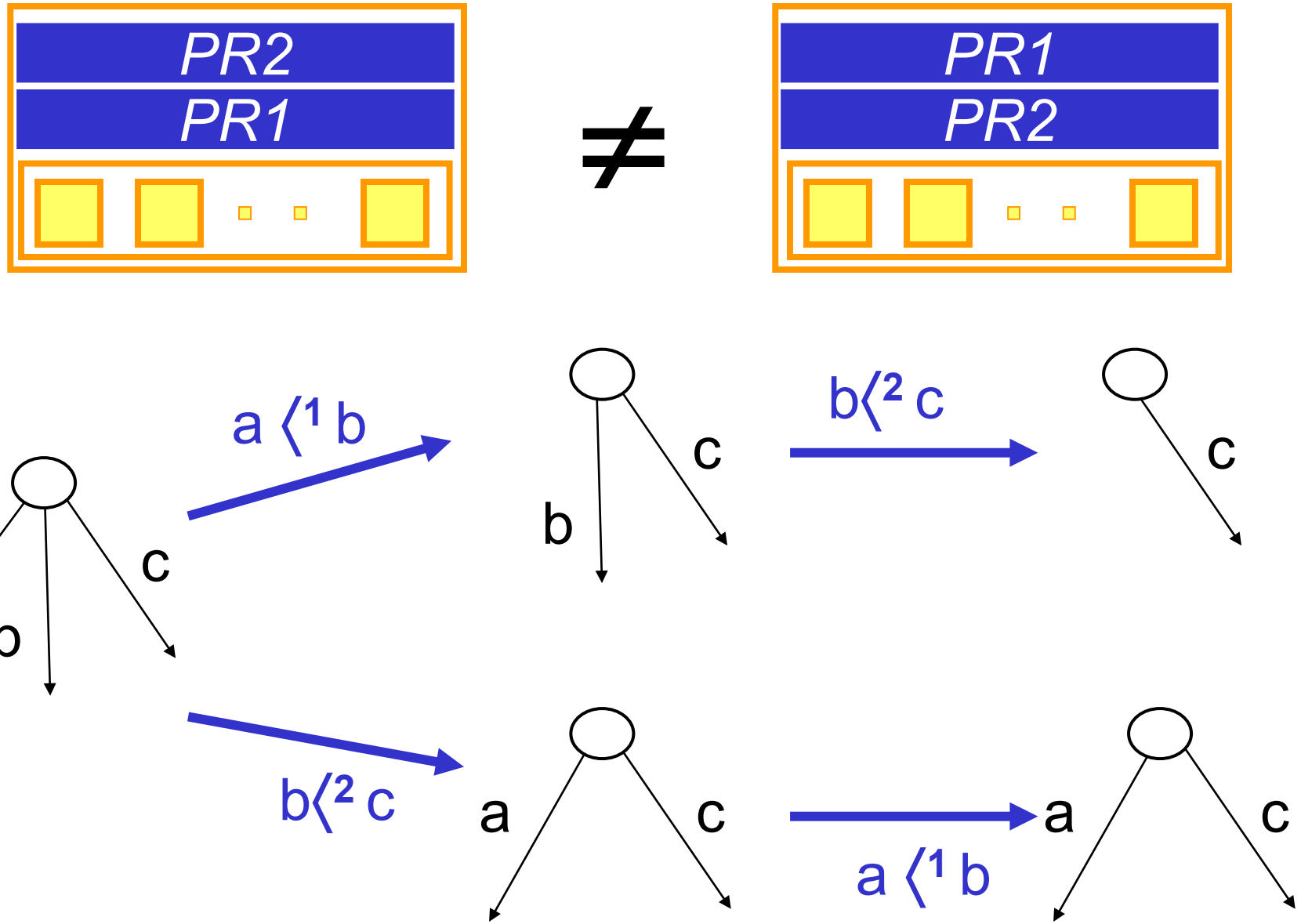


Modeling in BIP – Priorities: EDF policy

PR: D1-t1≤ D2- t2 → b2⟨ b1 D2-t2< D1-t1 → b1⟨ b2

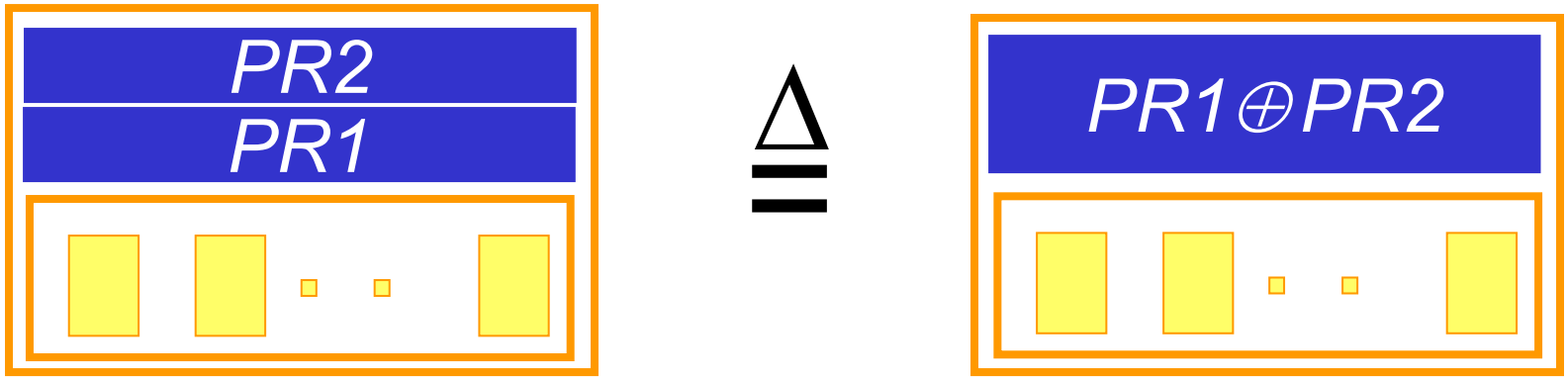


Modeling in BIP – Priorities: Composability



Modeling in BIP – Priorities: Composability

We take:



PR1⊕ PR2 is the least priority containing PR1∪PR2

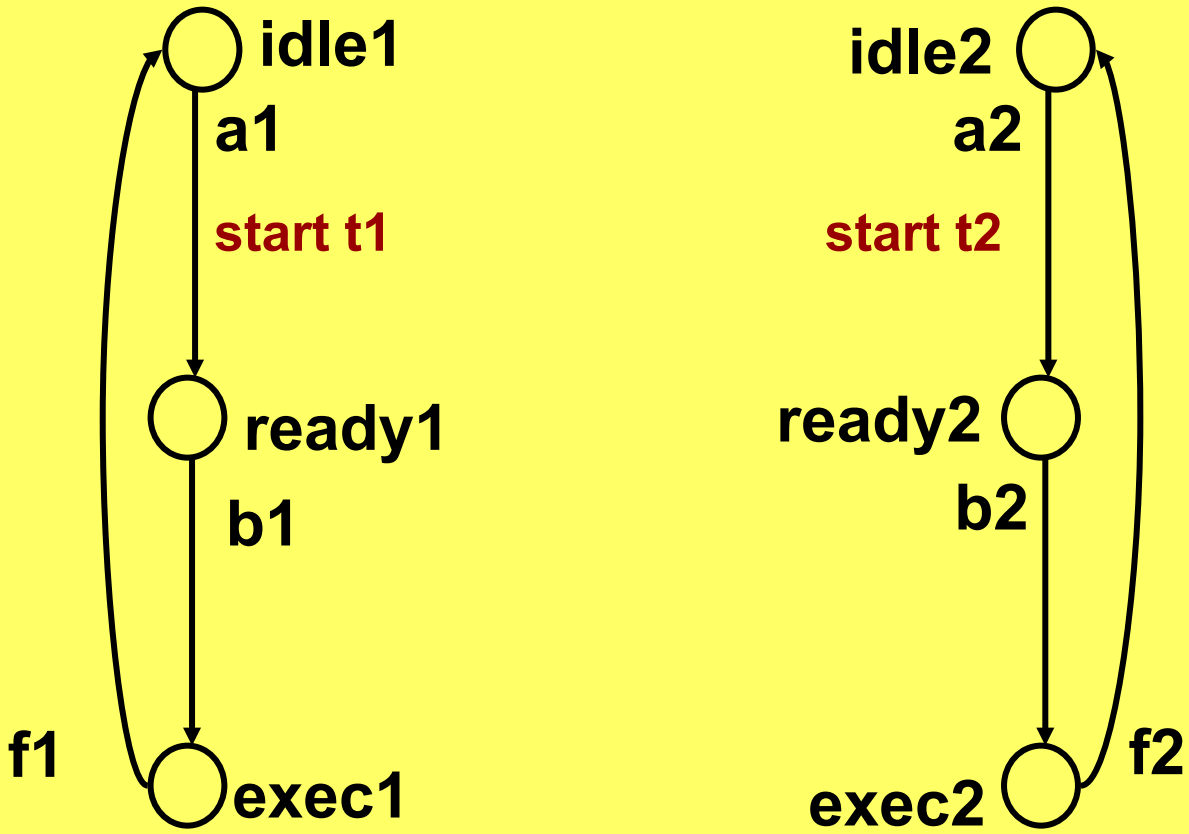
Results :

- The operation ⊕ is partial, associative and commutative
- PR1(PR2(B)) ≠PR2(PR1(B))
- PR1⊕ PR2(B) *refines* PR1∪PR2(B) *refines* PR1(PR2(B))
- Priorities preserve deadlock-freedom

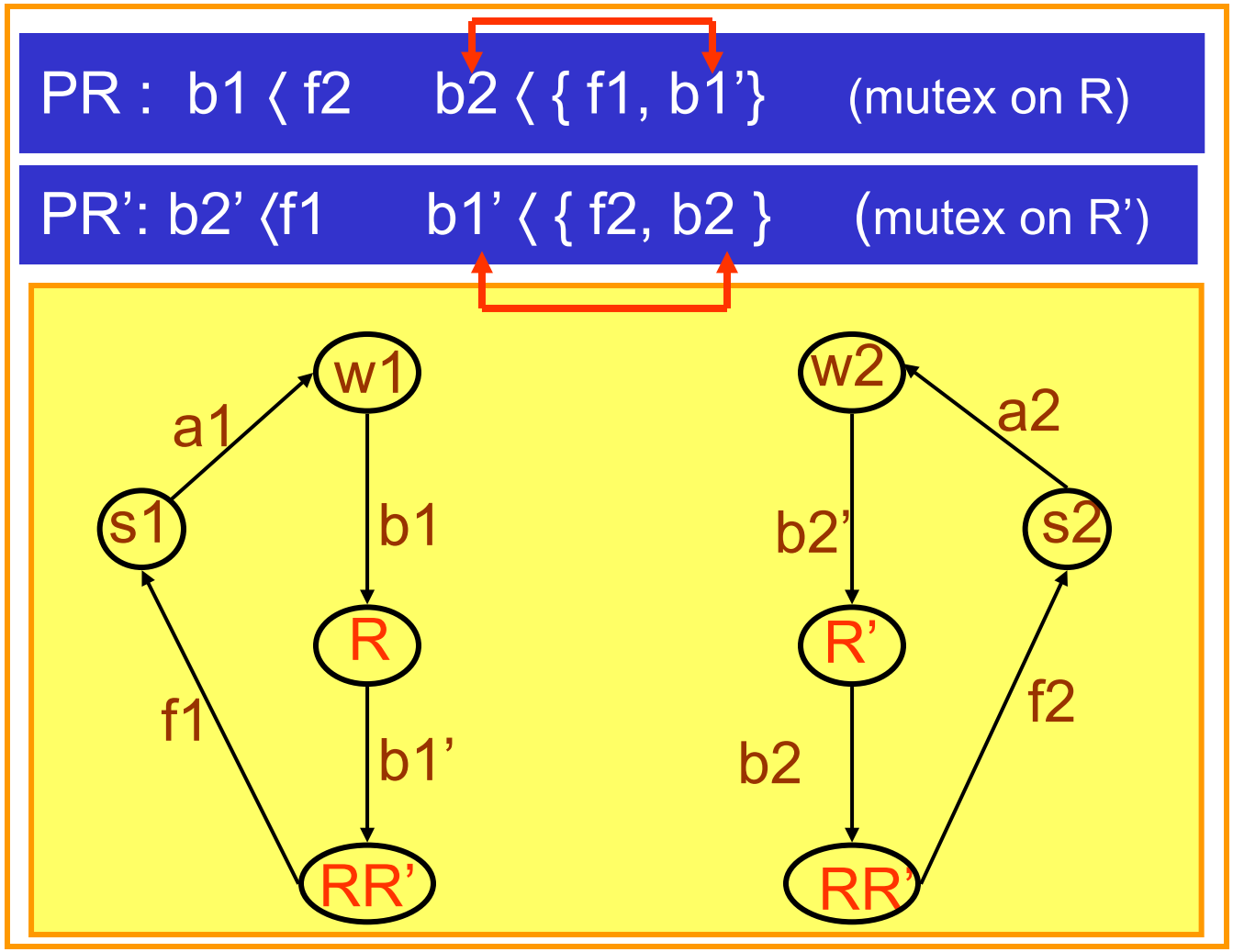
Modeling in BIP – Priorities: Mutual Exclusion + FIFO policy

$t1 \leq t2 \rightarrow b1\langle b2$ $\quad$ $t2 < t1 \rightarrow b2\langle b1$

$true \rightarrow b1\langle f2$ $\quad$ $true \rightarrow b2\langle f1$



Modeling in BIP – Priorities: Example



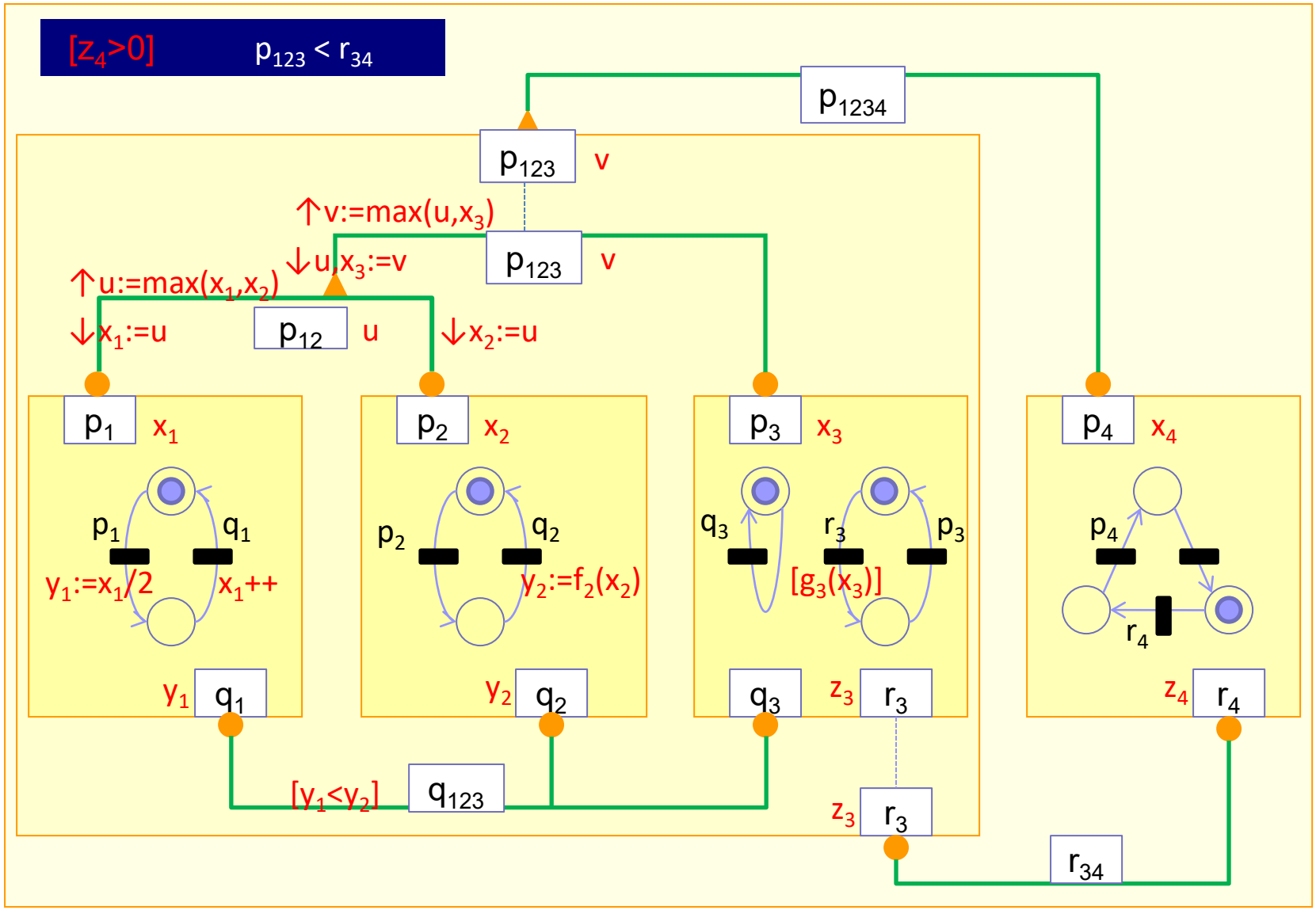
Risk of deadlock: PR⊕PR' is not defined

Modeling in BIP – The Language

Priorities

Interactions

Behavior



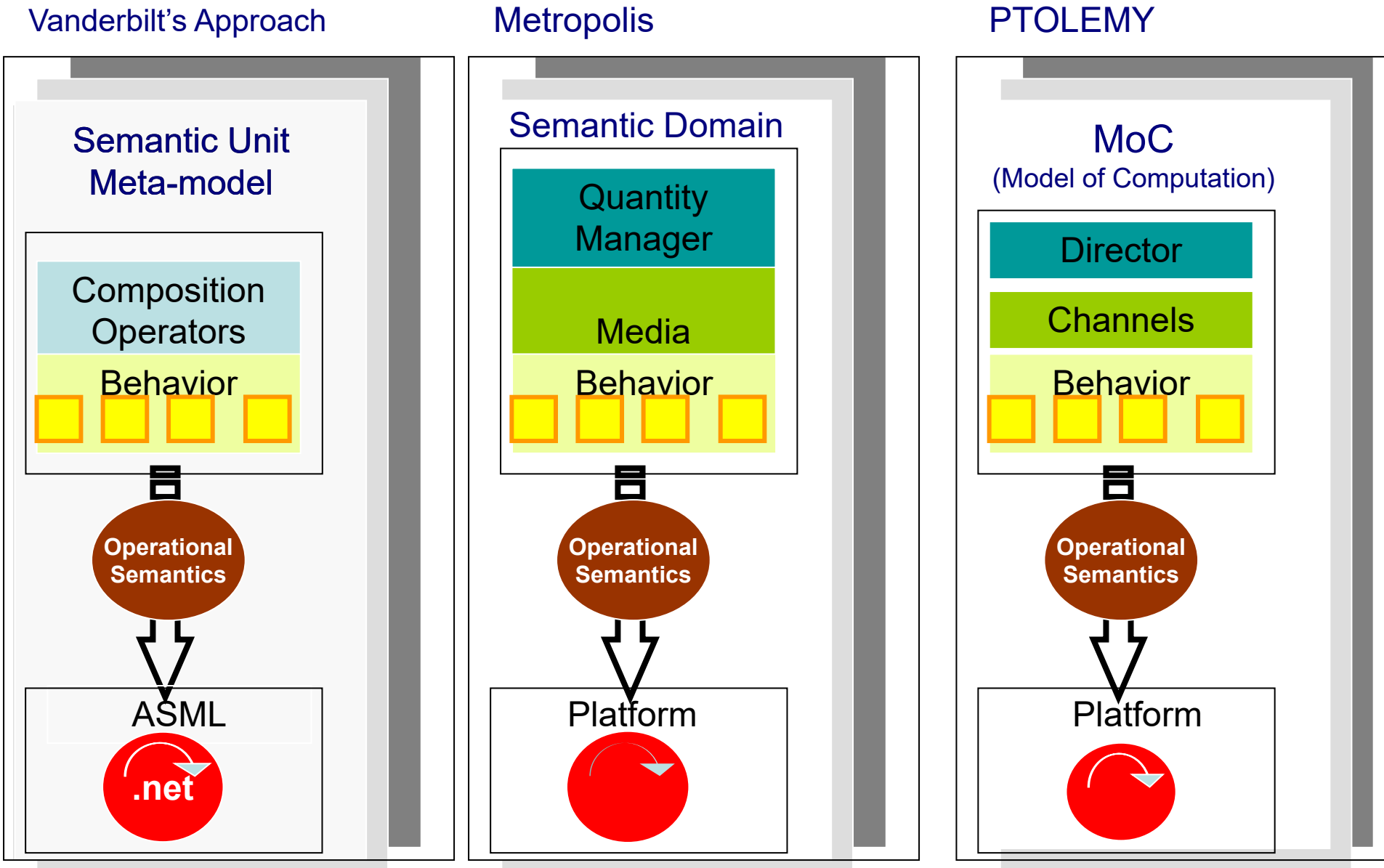
Modeling in BIP – The Language

```
// atomic component definitio

atomic type Atom(int p, int q
  data int x, y, z, …
  data DataType u, v, w, …
  port MyPort p1(x)
  port TypePort2 p2(y, u)

  place s1, s2, s3, s4, …

  initial to s1
    do { /* initialization co
  on p1 from s1 to s2
     provided guard1
     do { /* transition code
  on p2 from s2 to s3
     provided x < y
     do { {# plain C code #}
  …
  export port MyPort p1 is r1

end
```

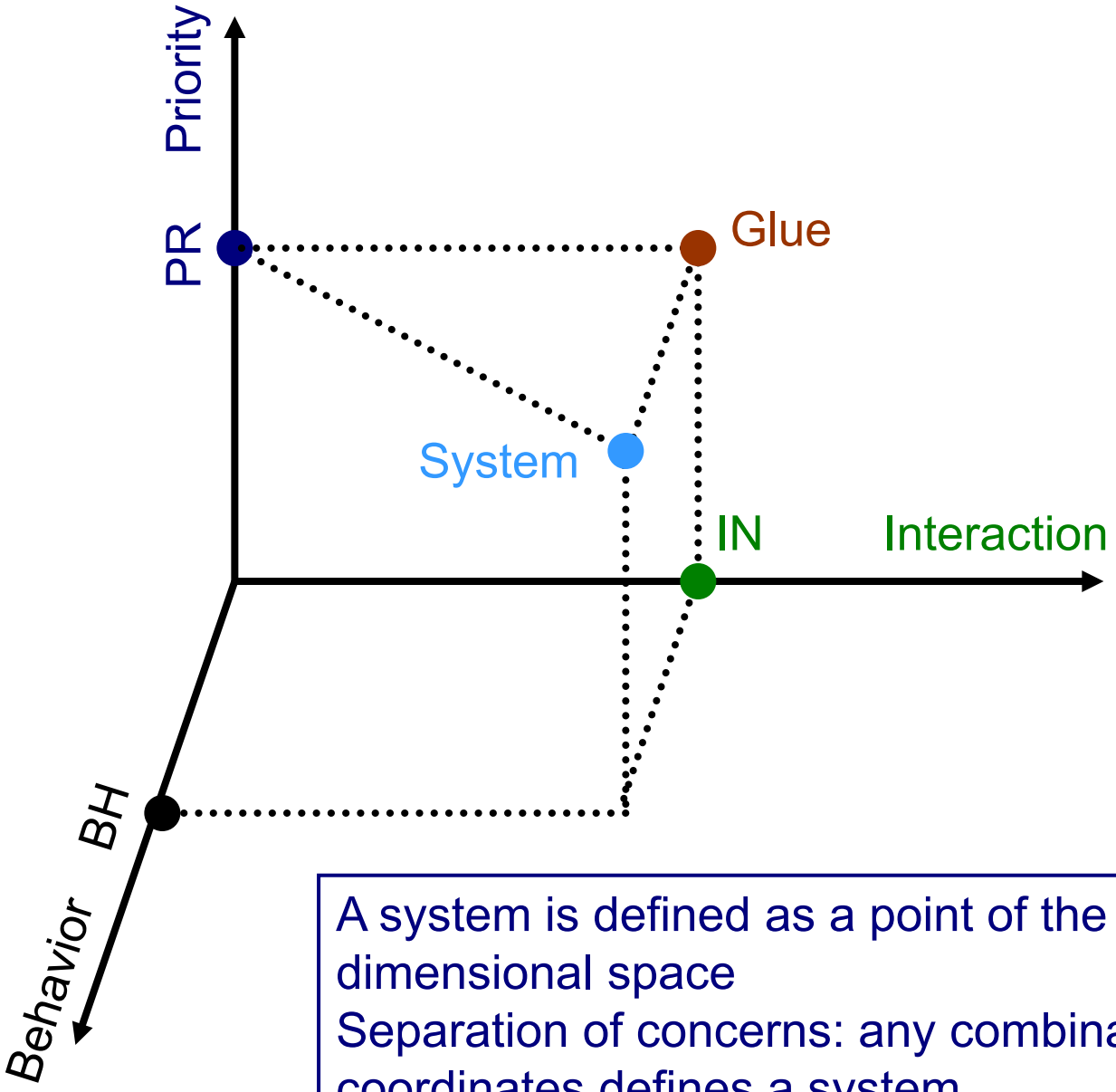
```
// connector type definition

connector type Bus (PortType1
                    PortType2
  define port-expression
  data int y
  …
  on interaction1 provided gu

     up { /*interaction code
     down { /* interaction co
  …
  on p1 p2 provided p1.x > 0
    up {y = p1.x + p2.x }
    down { {# p1.x = p2.x = y
  …
  export port PortType  p0(y)

end
```

```
// compound component type definition

compound type Compo(int p, …)
  component CompType_1 c1(p, …)
  …
  component CompType_n cn

  connector ConType_1 x1( c1.p, … c2.q )
  …
  connector ConType_k xk( x1.p0, cn.r )

  priority prio1
     provided guard
     xi:interaction1 < xj:interaction2
  …
  export port PortType1 c1.p is p
  export port PortTypek xk.p0 is q
    …
end
```

Modeling in BIP– Approaches encompassing heterogeneity

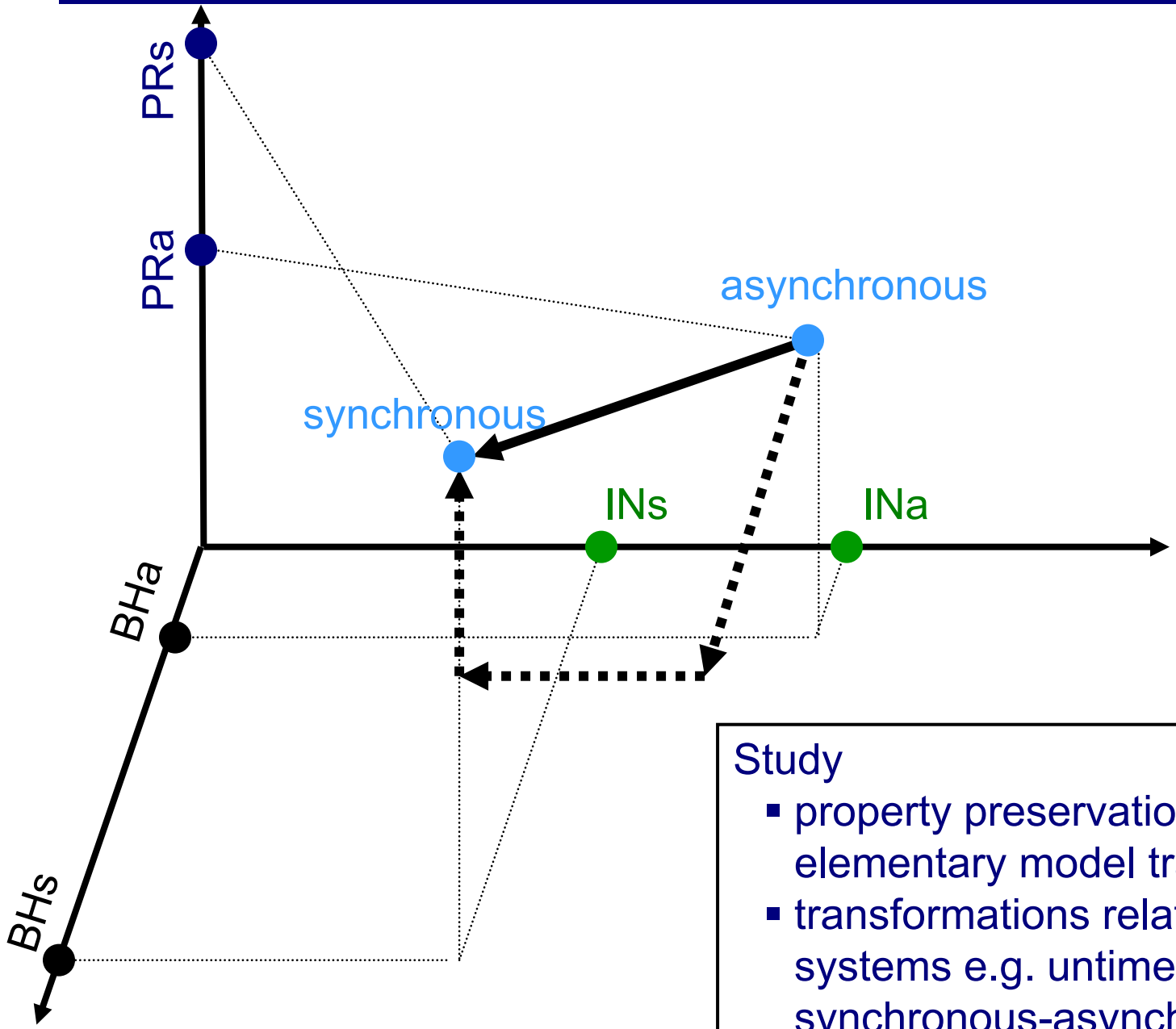


Modeling in BIP – System Construction Space



A system is defined as a point of the 3-dimensional space
Separation of concerns: any combination of coordinates defines a system

Modeling in BIP – System Construction Space



Study

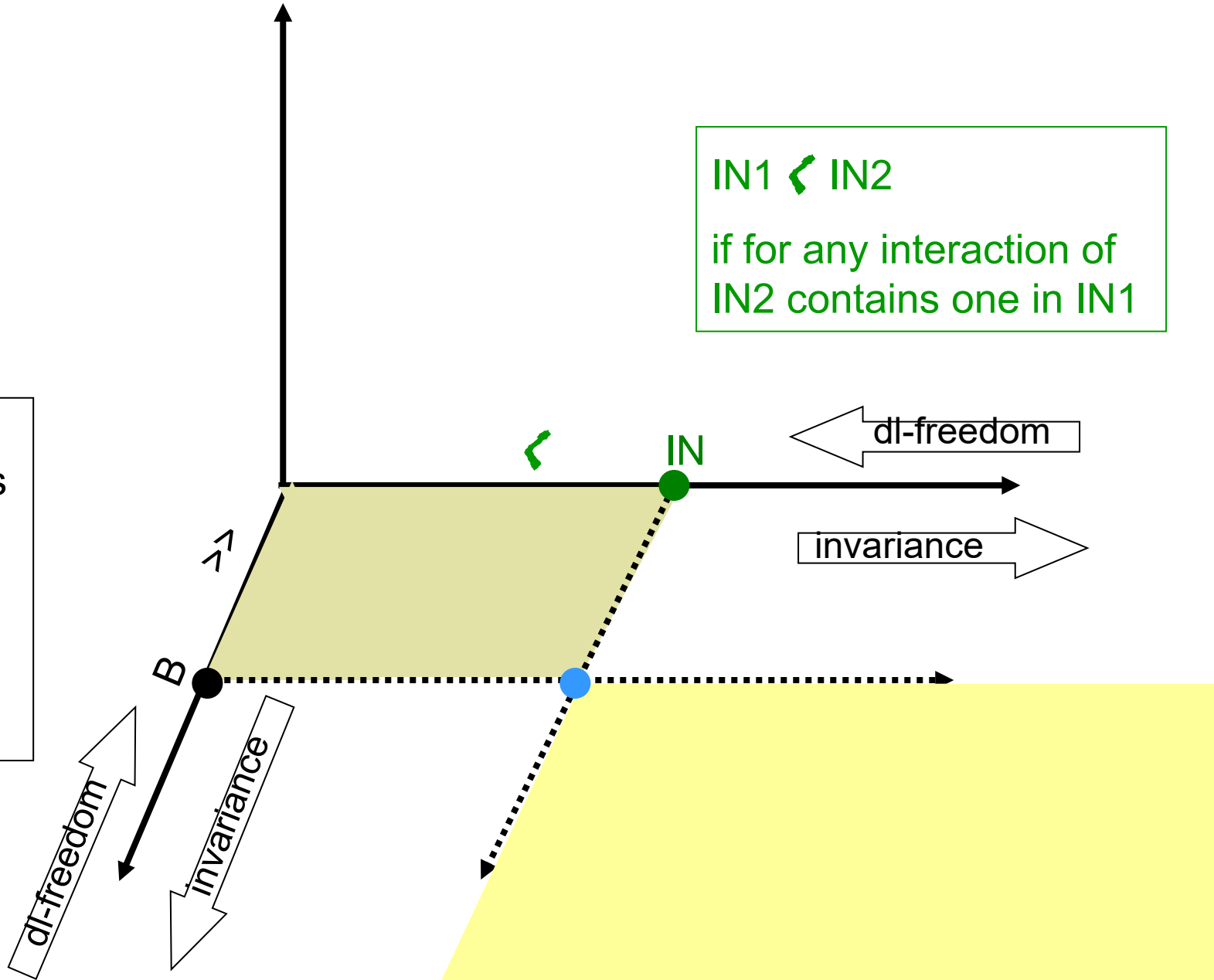
- property preservation results by elementary model transformations
- transformations relating classes of systems e.g. untimed-timed, synchronous-asynchronous

Modeling in BIP– System Construction Space: Incrementality

IN1 ≮ IN2

if for any interaction of IN2 contains one in IN1

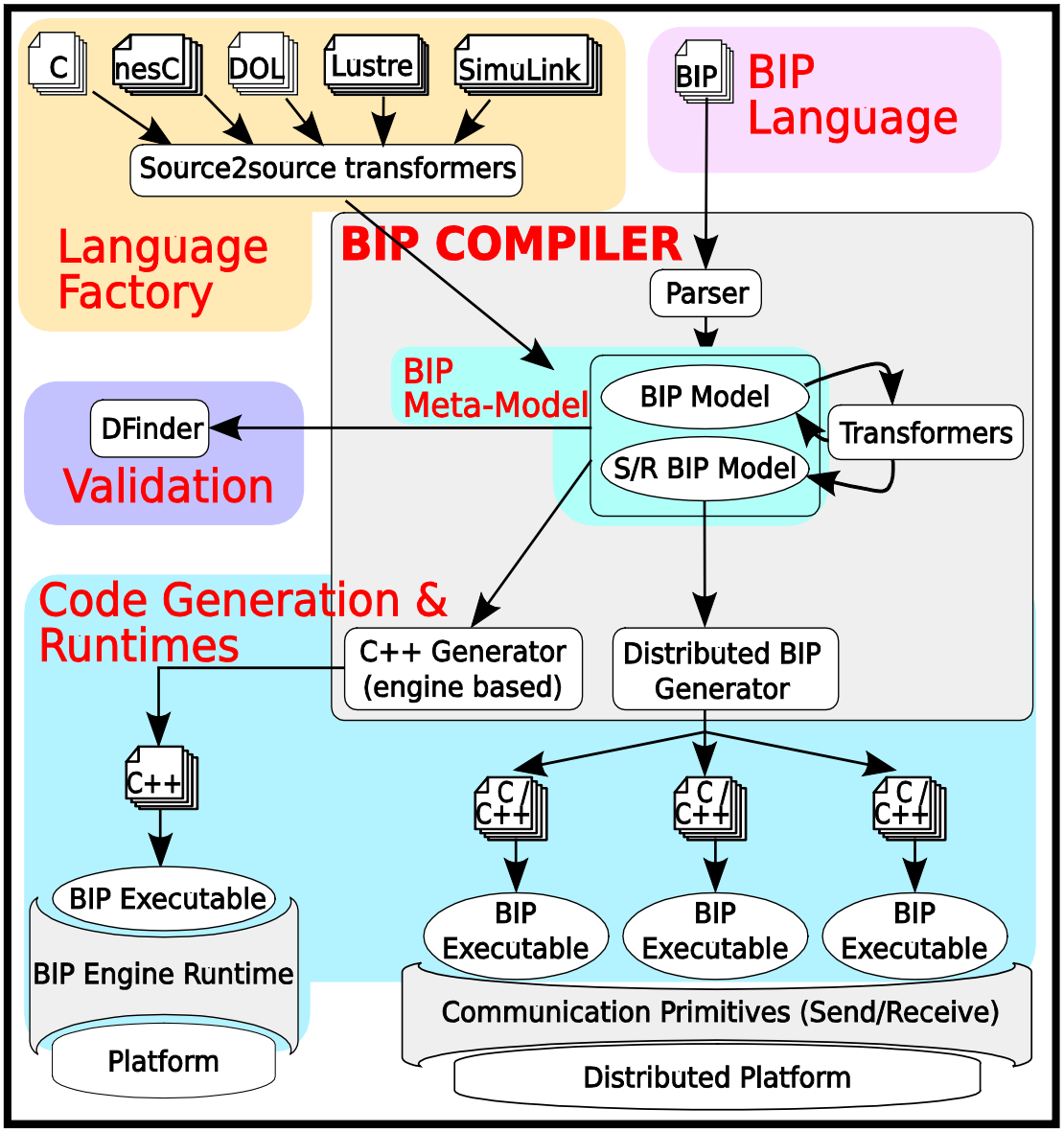
B1<<B2 if B1 simulates B2 and the simulation relation is total



- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

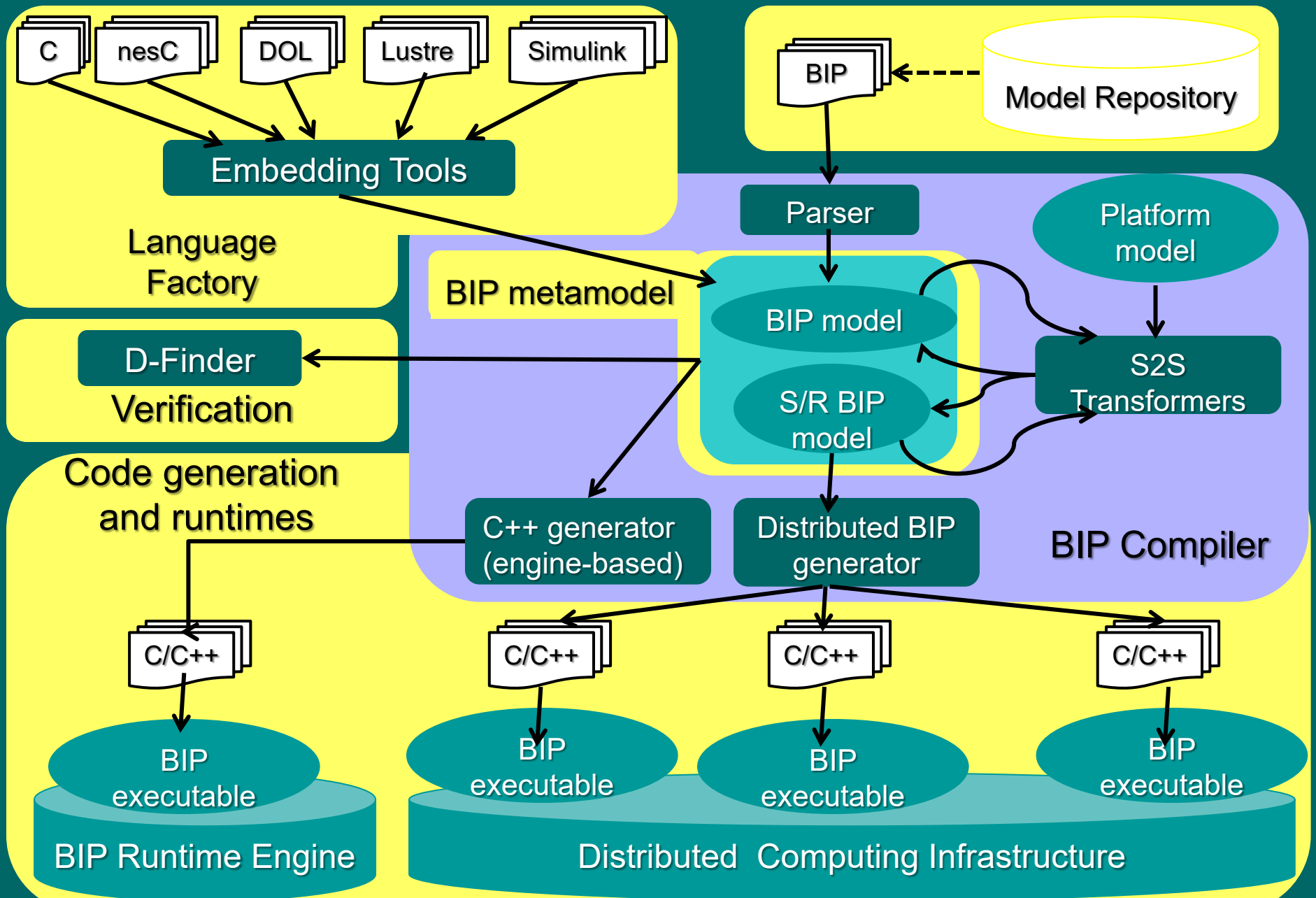
OVERVIEW

The BIP Toolset – Overview

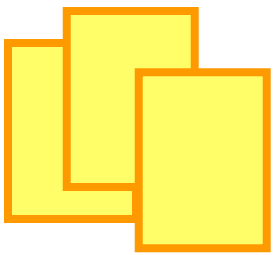


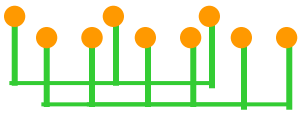
http://www.bip-components.com

The BIP Toolset – Overview



The BIP Toolset – The Execution Engine





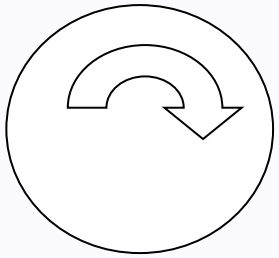
C→a⟨b

BIP model

Component Meta-model

Interaction Meta-model

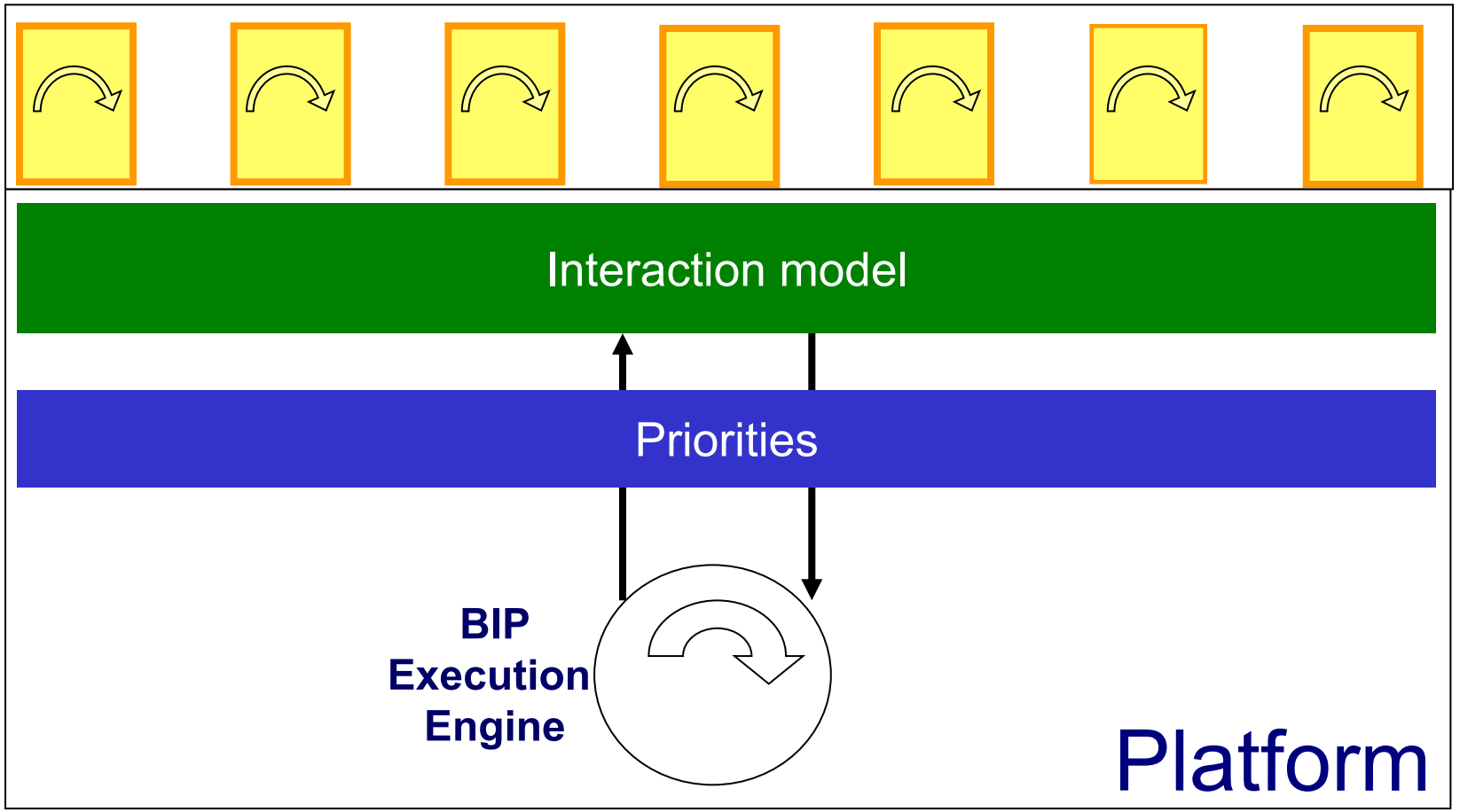
Dynamic priorities Meta-model



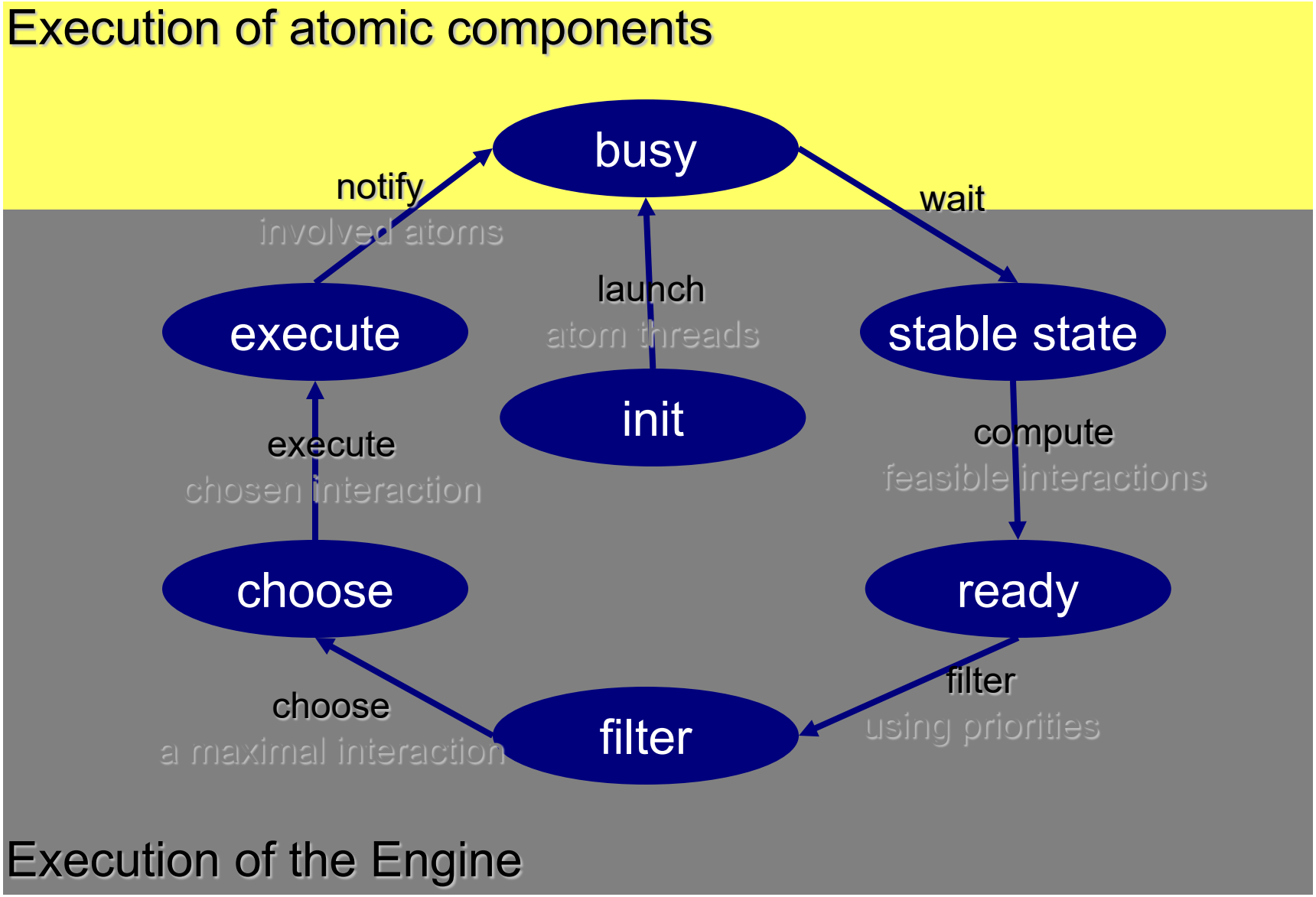
BIP Execution Engine

Platform

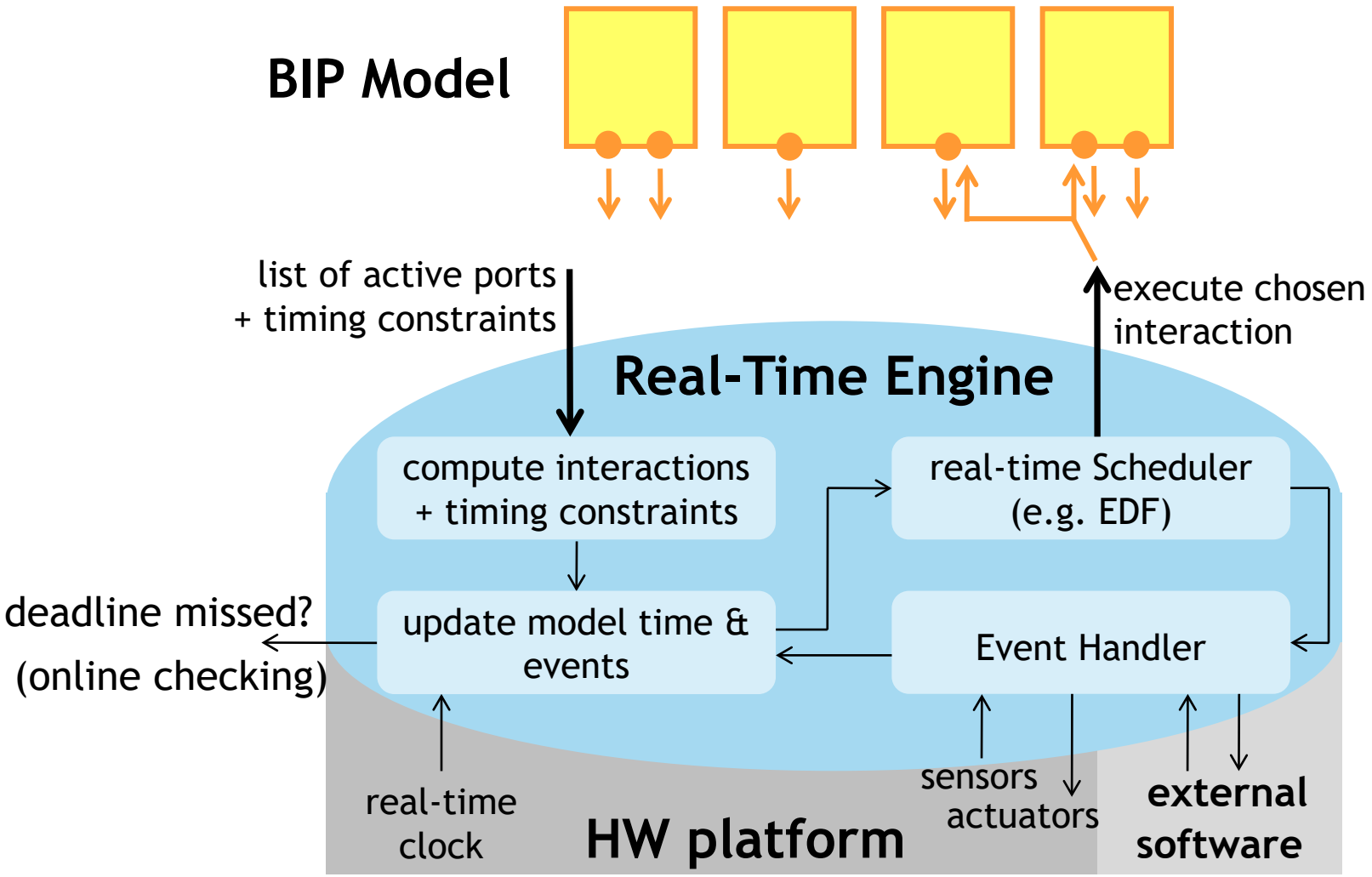
The BIP Toolset – The Execution Engine



The BIP Toolset – The Execution Engine



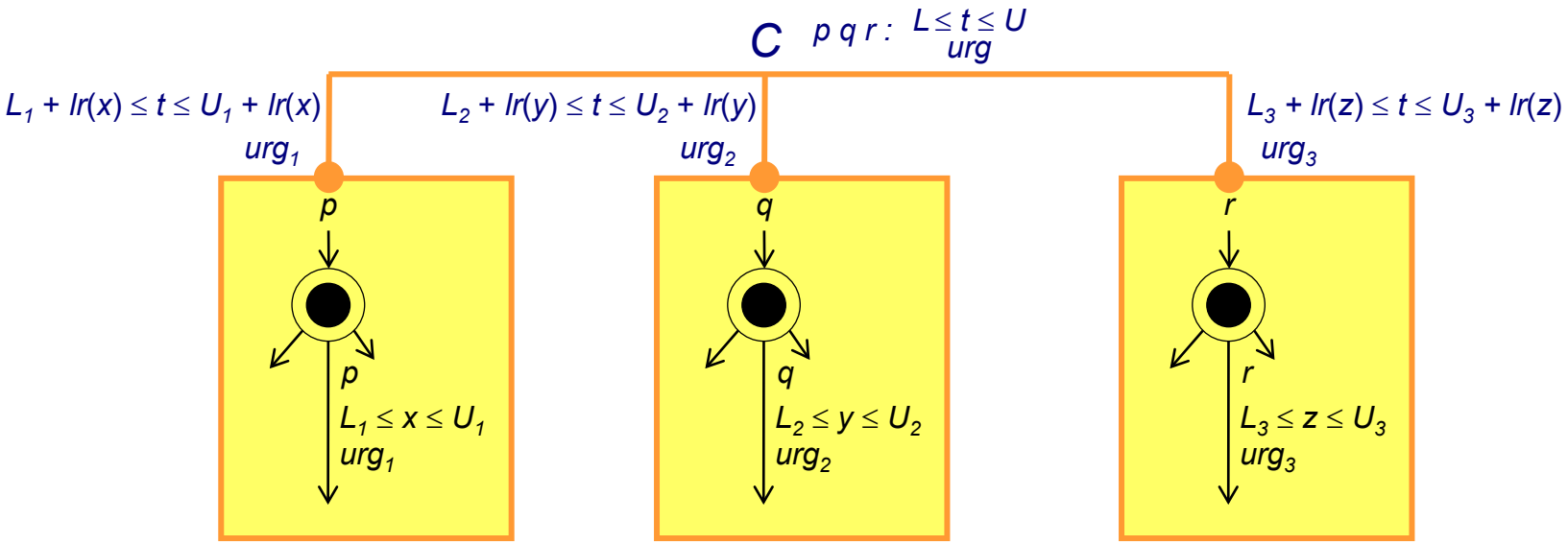
The BIP Toolset – The RT Execution Engine



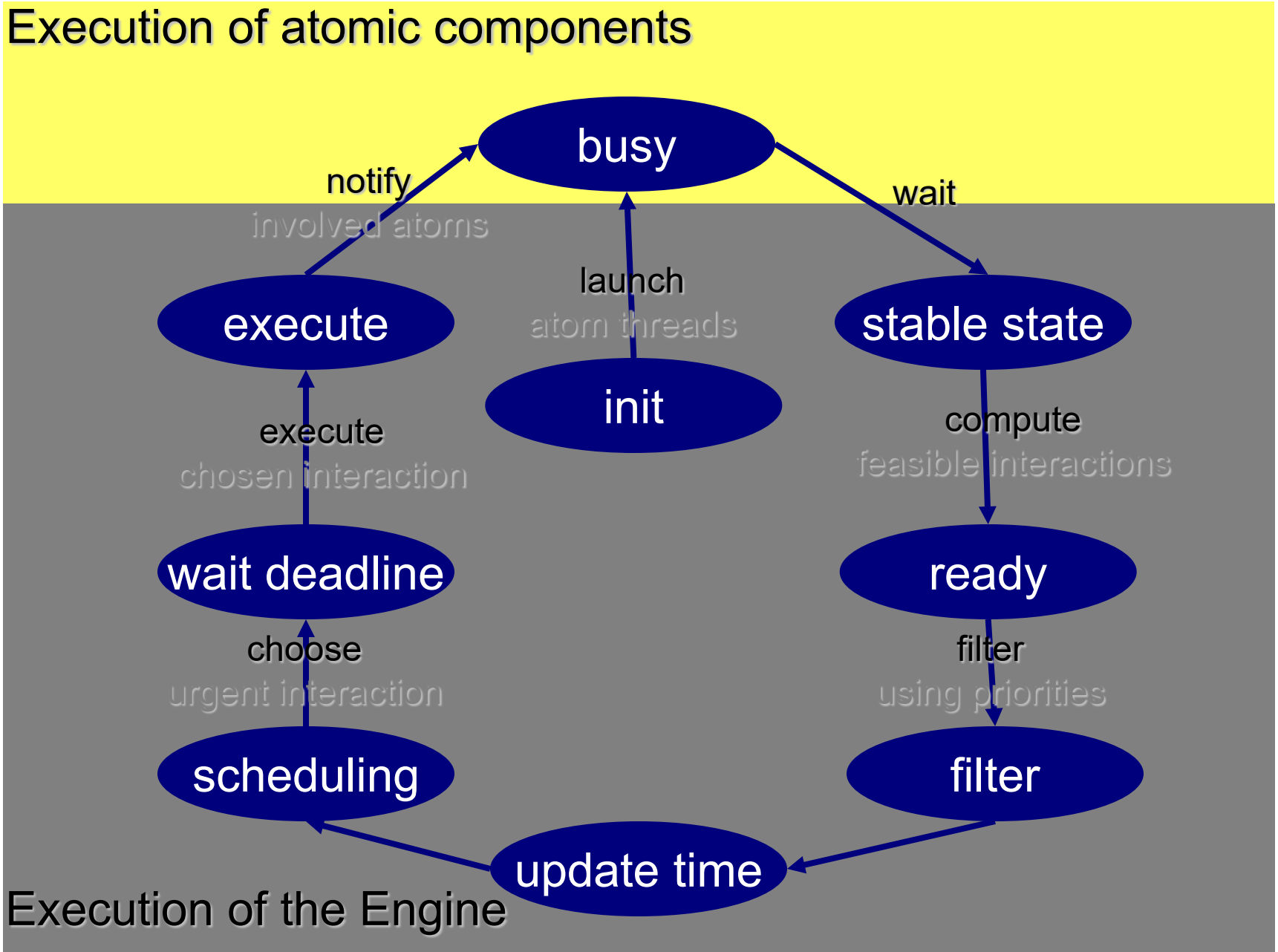
The BIP Toolset – The RT Execution Engine

- Conversion of local timing constraints using a single global clock *t*:
  - *t* is never reset, i.e. it represents the total time elapsed
  - for a clock *x*, *tr*(*x*) corresponds to the value of *t* at the instant of the lasts reset of *x*.

- Resulting constraint and urgency is $L \leq t \leq U$ *urg*, where:
  - $L = \max L_i$
  - $U = \min U_i$
  - *urg* = max urg_i (lazy < delayable < eager).



The BIP Toolset – The RT Execution Engine

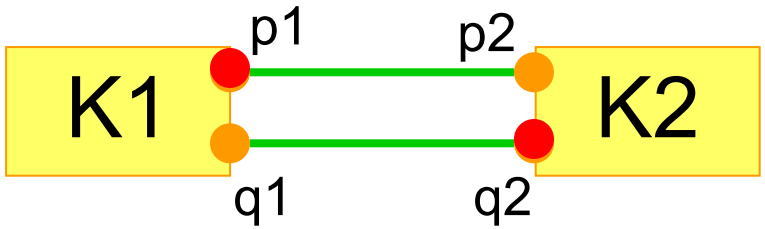


- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

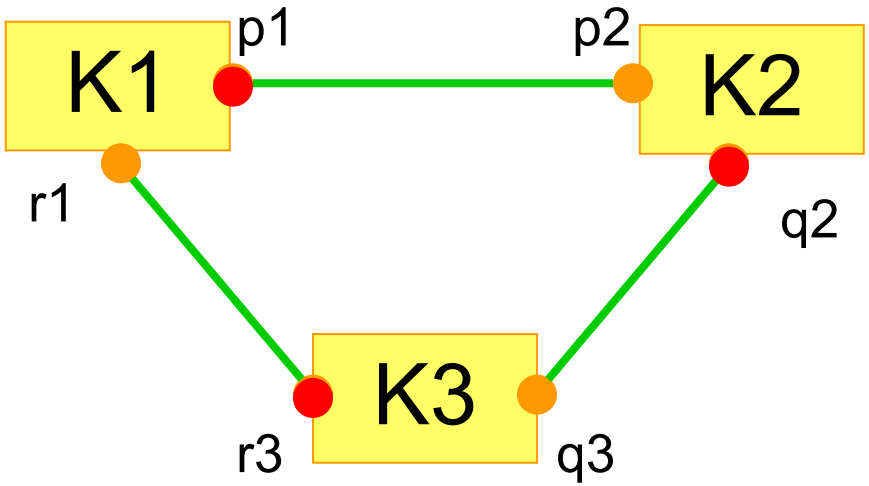
OVERVIEW

Compositional Verification

Verify global deadlock-freedom of a system
by separate analysis of the components and of the architecture.



Potential deadlock
$D = en(p1) \wedge \neg\, en(p2) \wedge en(q2) \wedge \neg\, en(q1)$

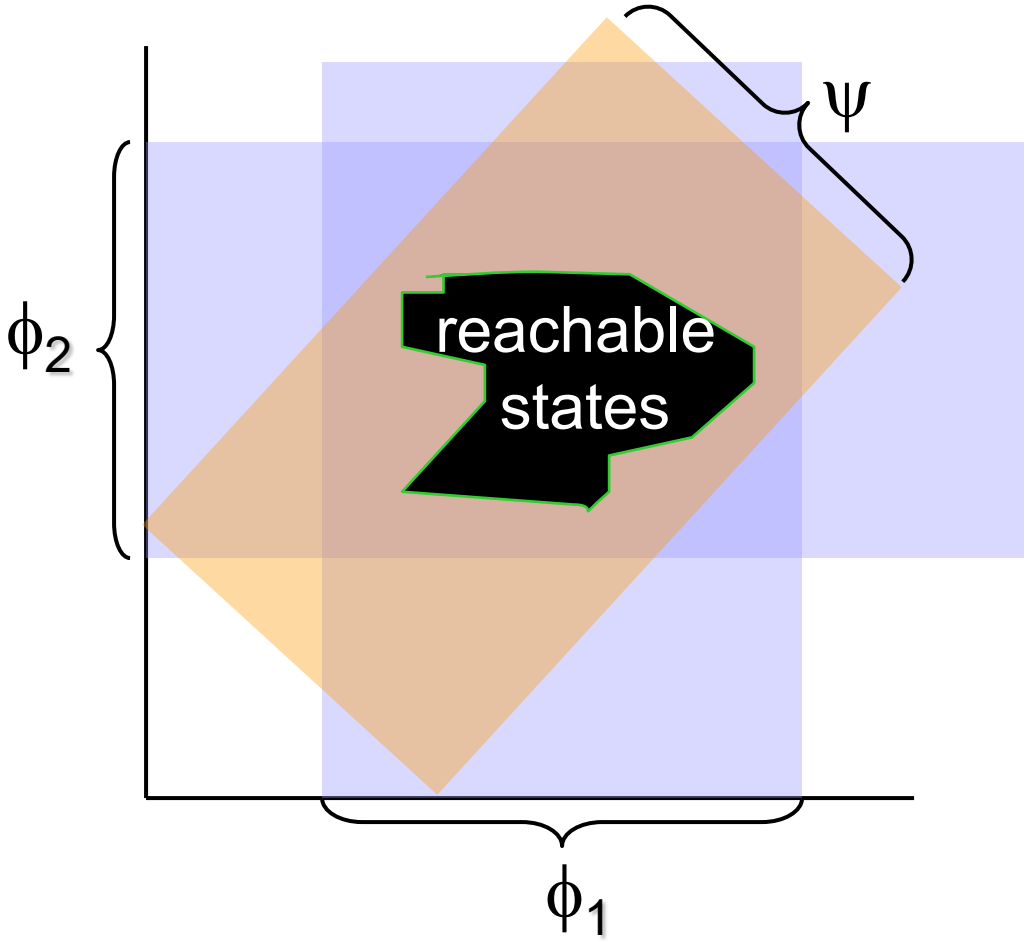


Potential deadlock
$D = en(p1) \wedge \neg\, en(p2) \wedge en(q2) \wedge \neg\, en(q3) \wedge en(r3) \wedge \neg\, en(r1)$

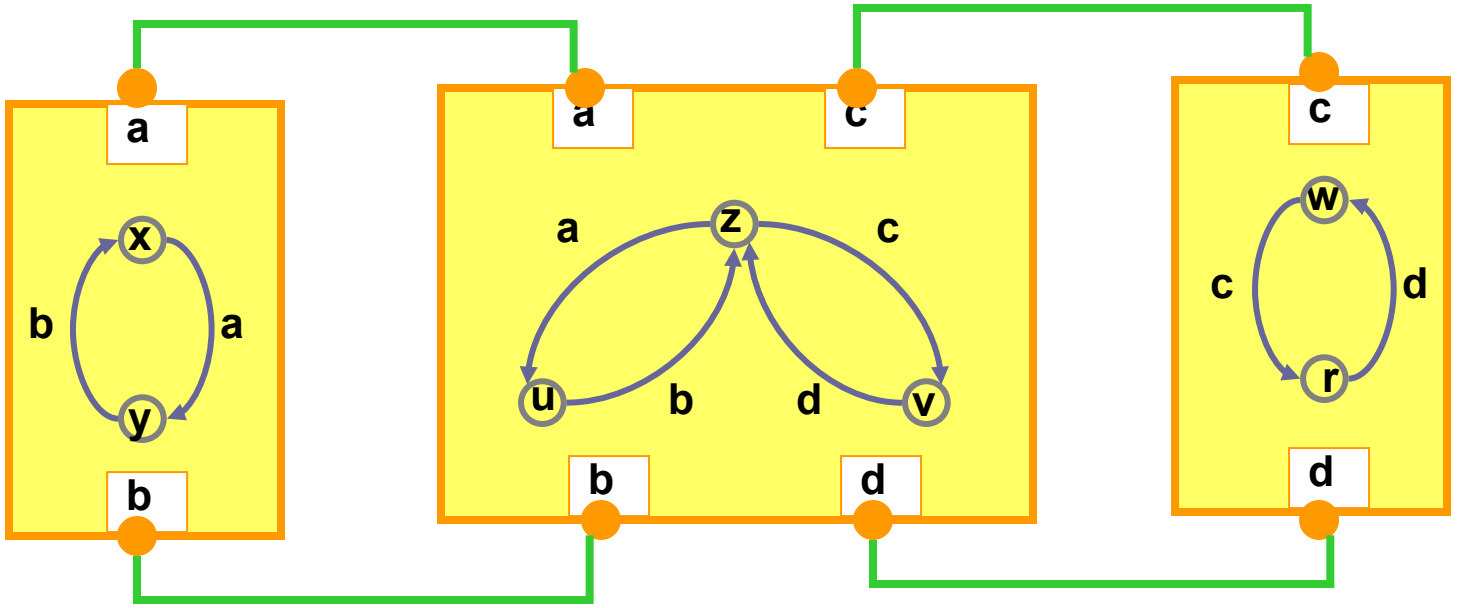
Compositional Verification – Interaction Invariants

$$\frac{B_1 \models \Box\,\phi_1 \quad B_2 \models \Box\,\phi_2 \quad \psi \in \Pi(\gamma(B_1, B_2),\phi_1,\phi_2) \quad \phi_1 \wedge \phi_2 \wedge \psi \Rightarrow \chi}{\gamma(B_1, B_2) \models \Box\chi}$$

Method:
Eliminate potential deadlocks D by computing compositionally global invariants χ such that $\chi \wedge D = \text{false}$



Compositional Verification – Interaction Invariants



$x \Rightarrow y \vee u$
$y \Rightarrow x \vee z$

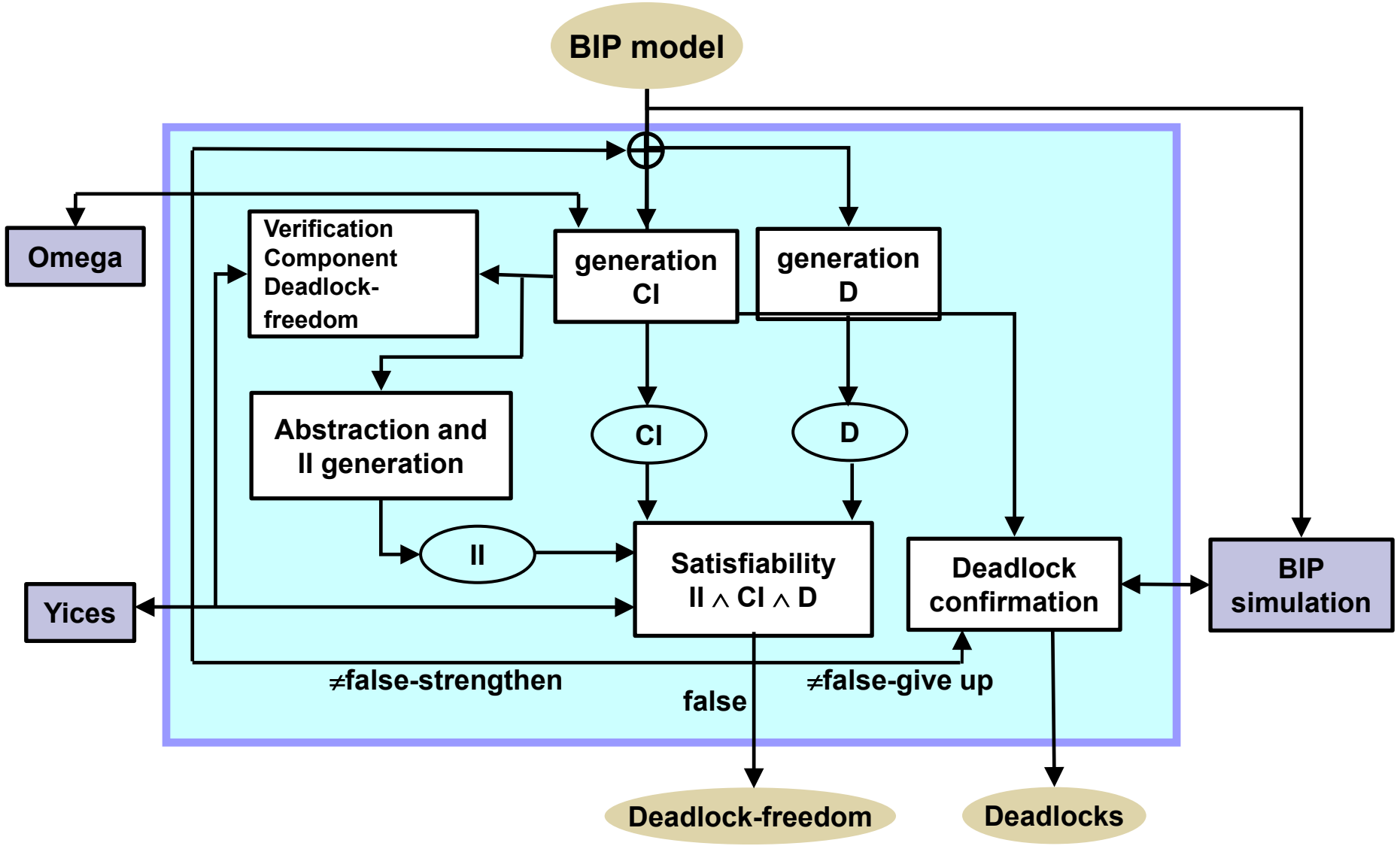
$z \Rightarrow (y \vee u) \wedge (v \vee r)$
$u \Rightarrow x \vee z$
$v \Rightarrow w \vee z$

$w \Rightarrow (v \vee r)$
$r \Rightarrow w \vee z$

Minimal solutions define invariants :

- Component invariants: $x \vee y$, $z \vee u \vee v$, $w \vee r$
- Interaction invariants: $x \vee u$, $z \vee y \vee v$, $z \vee y \vee r$, $z \vee u \vee r$, $w \vee v$

Compositional Verification – D-Finder

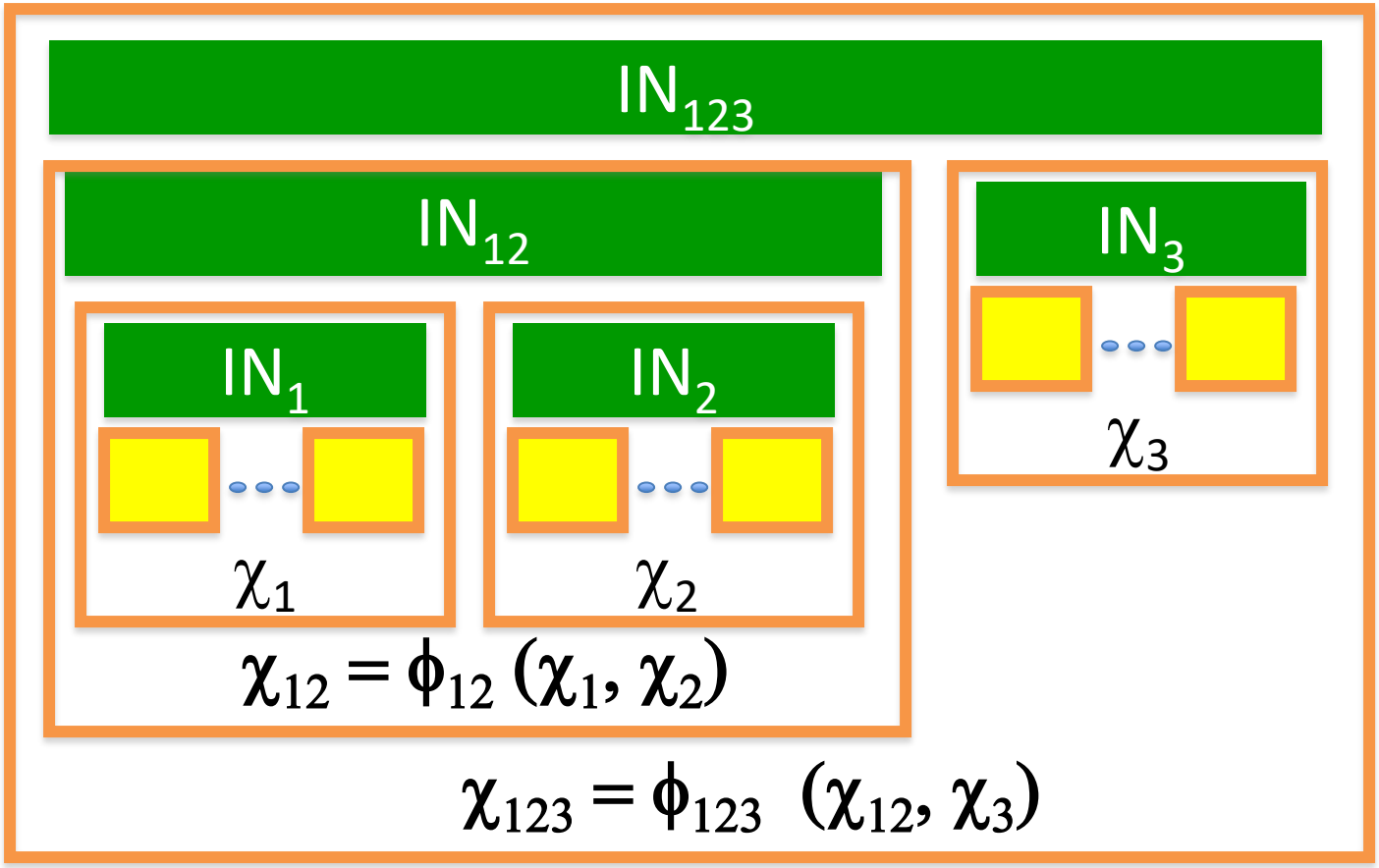


Compositional Verification – D-Finder

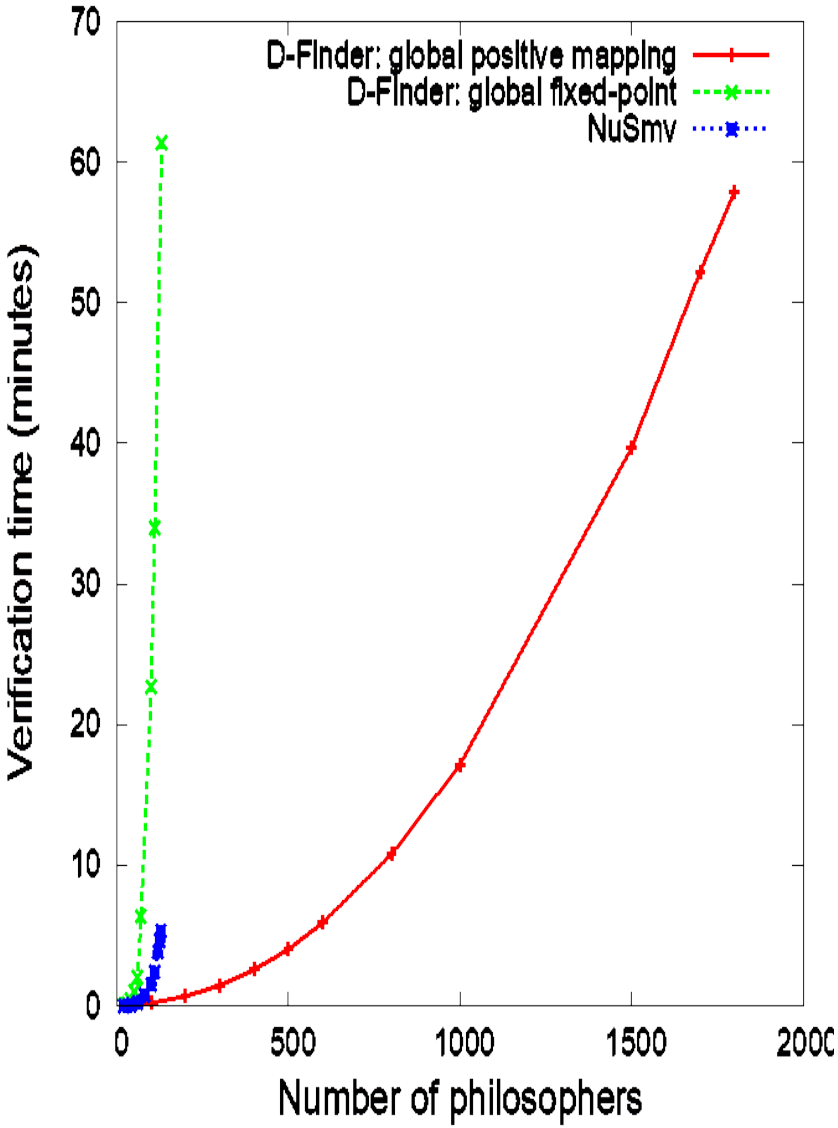
Example	Number of Comp	Number of Ctrl States	Number of Bool Variables	Numb of Int Var	Number Potential Deadlocks	Number Remaining Deadlocks	Verification Time
Temperature Control (2 rods)	3	6	0	3	8	3	3s
Temperature Control (4 rods)	5	10	0	5	32	15	6s
UTOPAR (40 cars,256 CU)	297	795	40	242	--	0	3m46s
UTOPAR (60 cars, 625 CU)	686	1673	60	362	--	0	25m29s
R/W(10000 readers)	10002	20006	0	1	--	0	36m10s
Philosophers (13000)	26000	65000	0	0	--	3	38m48s
Philosophers (10000)	20000	50000	0	0	--	3	29m30s
Smokers (5000)	5001	10007	0	0	--	0	14m
Gas stations (500 pumps, 5000 customers)	5501	21502	0	0	--	0	18m55s

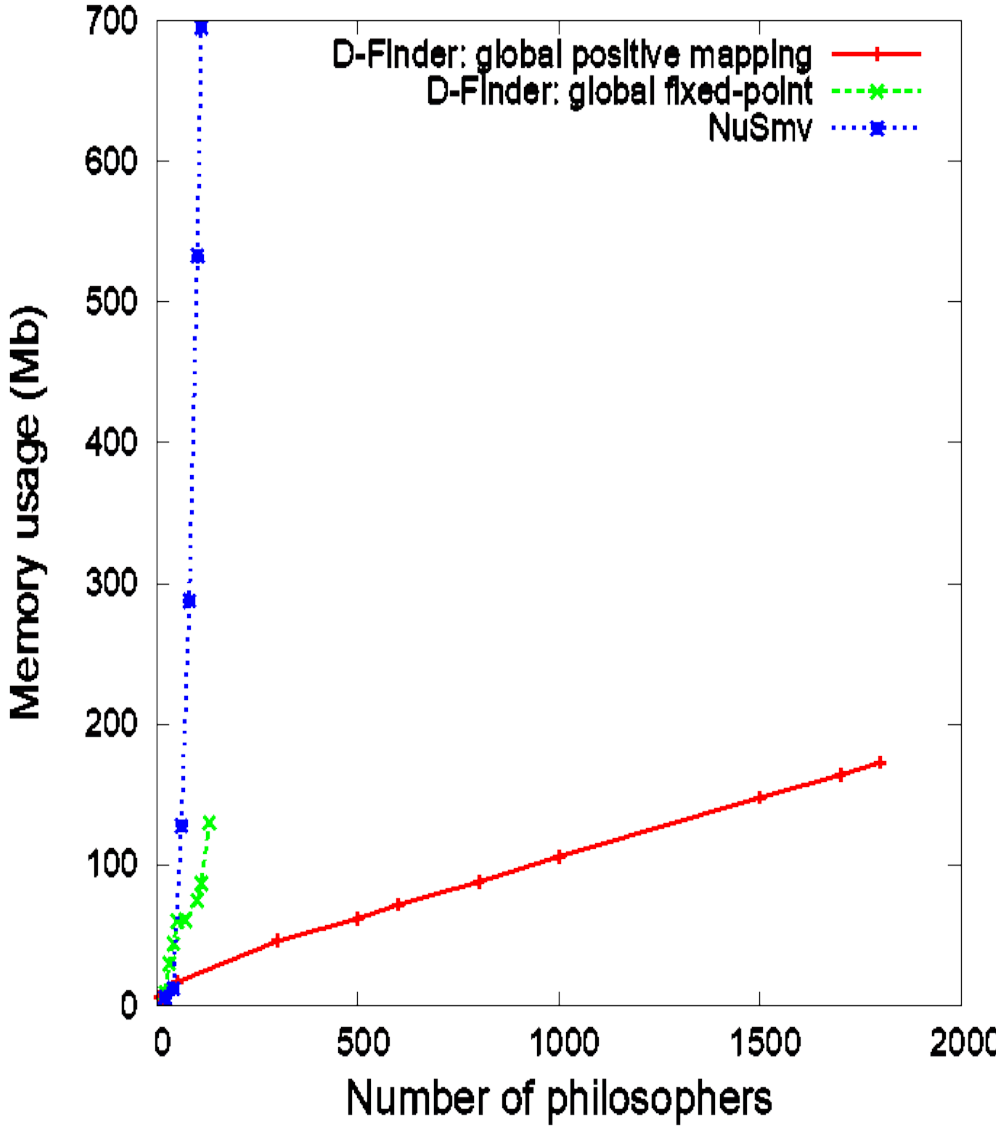
Results obtained by using the D-Finder tool: http://www-verimag.imag.fr/~thnguyen/tool/

Compositional Verification – Incremental Verification

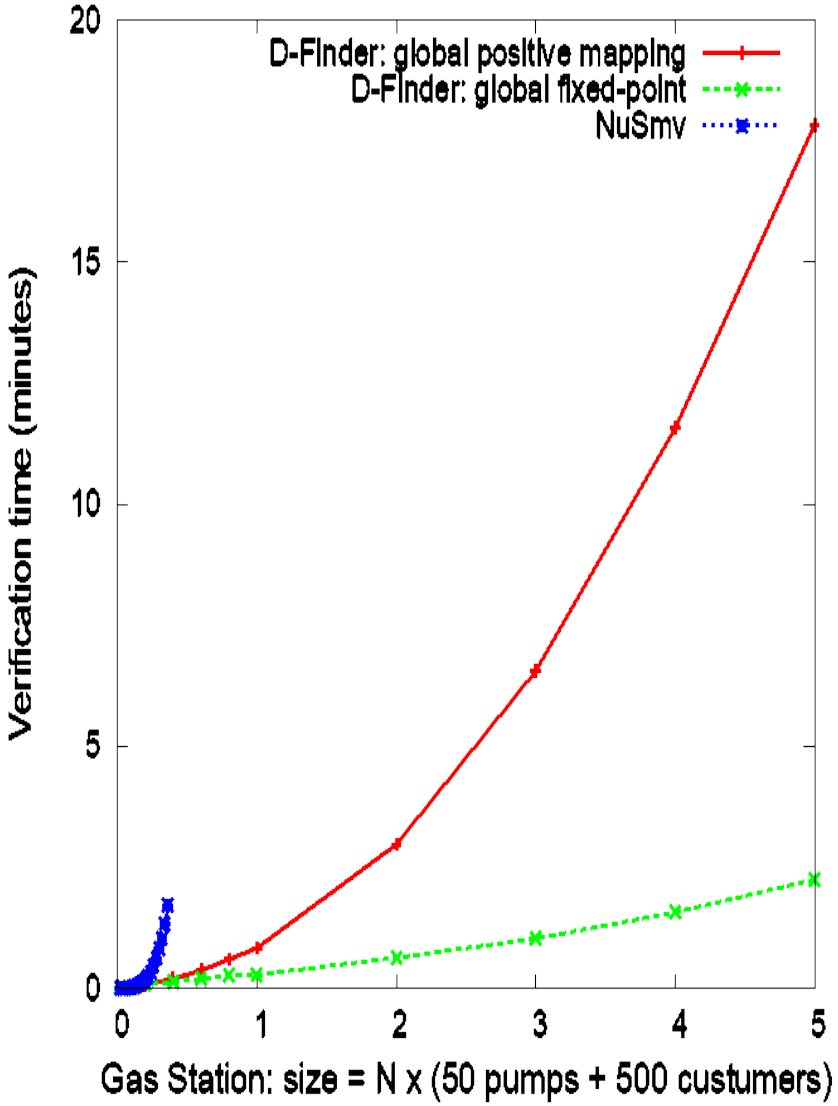


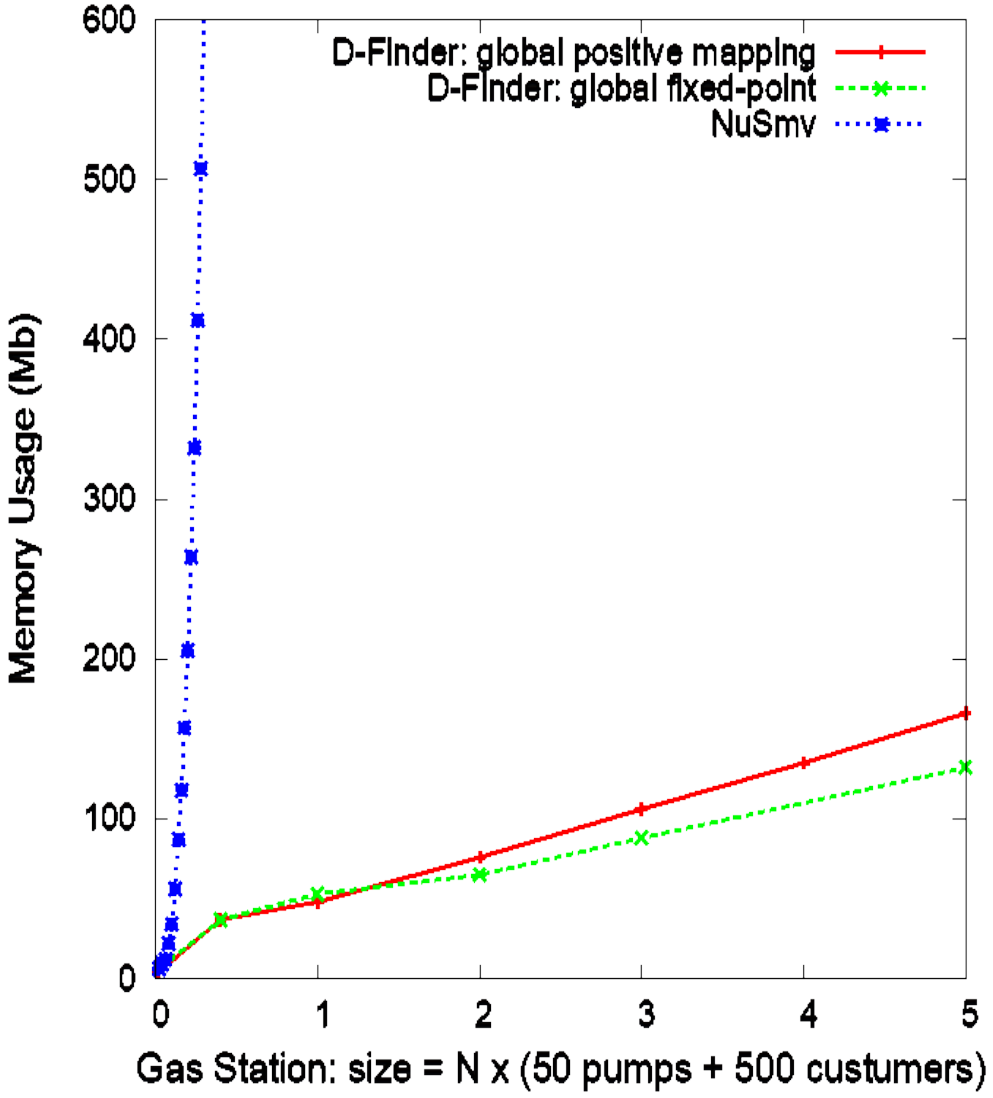
Compositional Verification – D-Finder





Compositional Verification – D-Finder



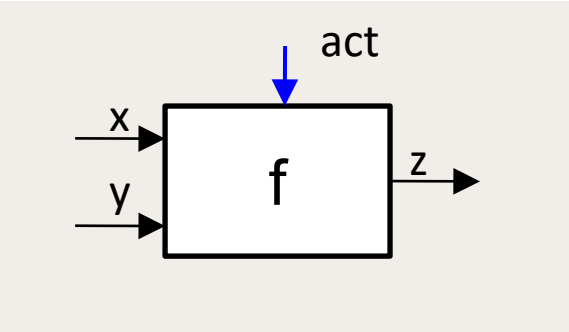


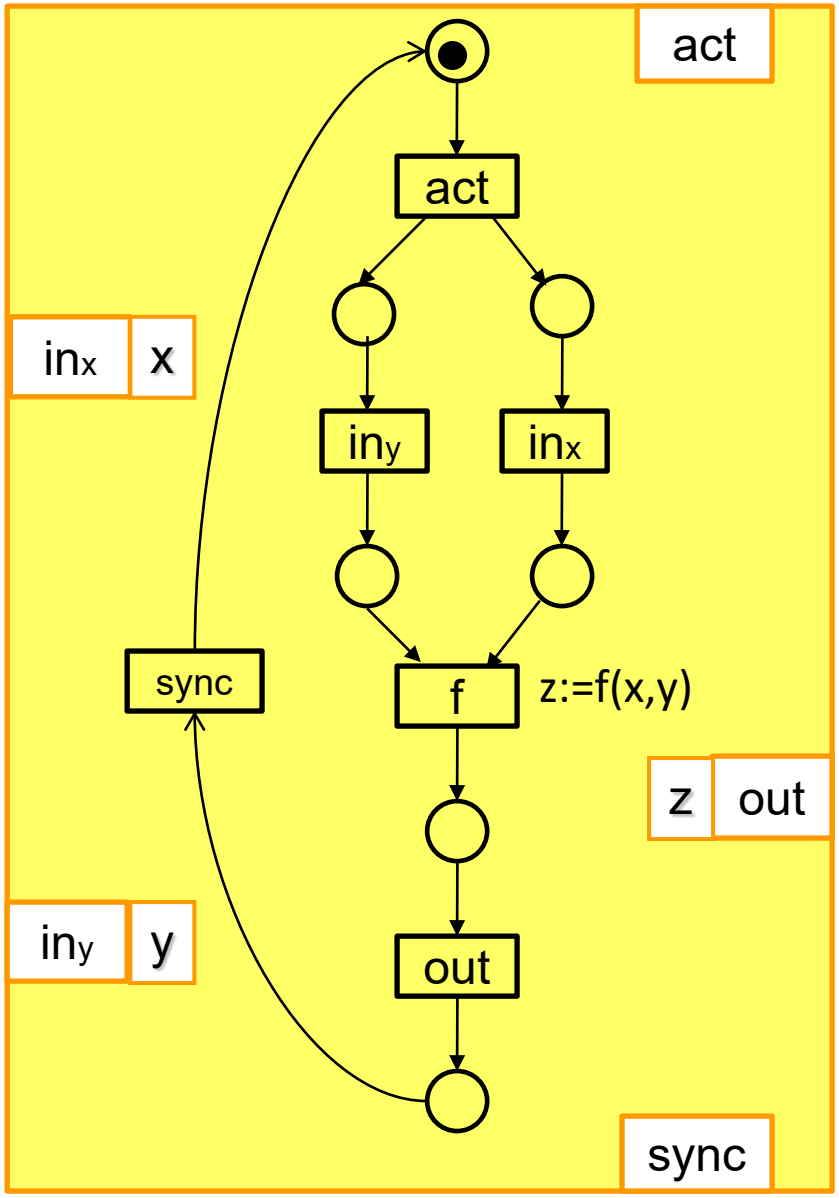
- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Synchronous Systems – Automata based Model

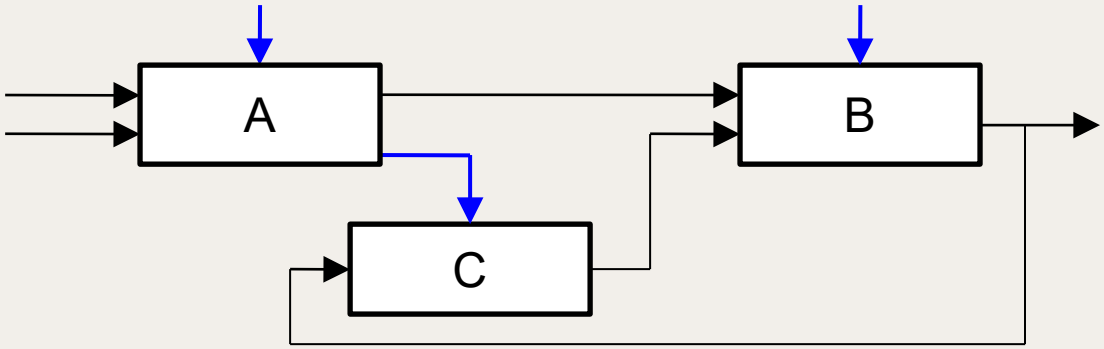
Idea: Represent synchronous components as atomic BIP components

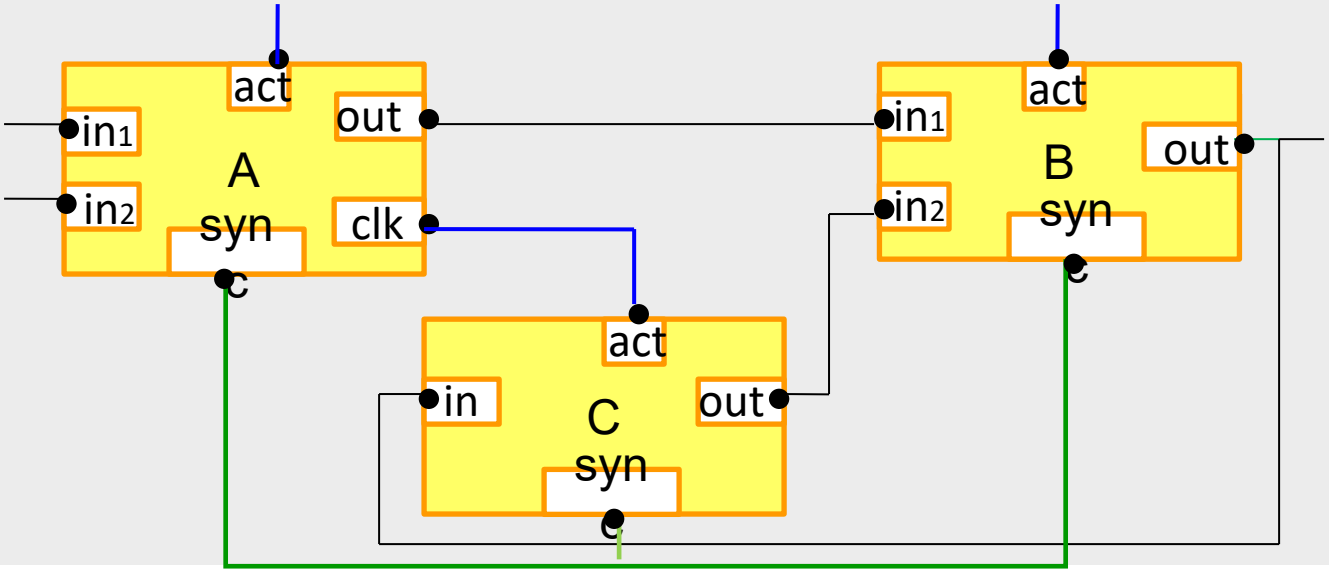




- In **each step** the inputs *x* and *y* are updated and an output *z* is produced
- The **sync** transition denotes the end of a step

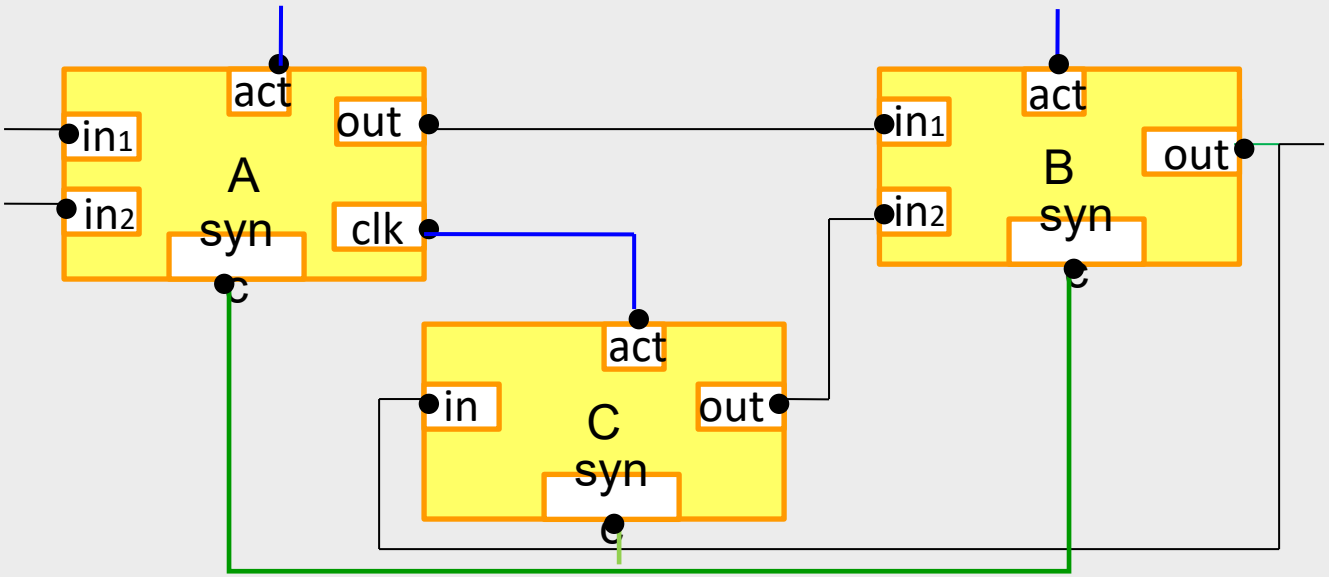
Synchronous Systems – Automata based Model





- Synchronous components are replaced by **automata based models**
- **Data-flow** interactions between data ports replace data-flow links
- **Control** interactions replace activation links
  - The **sync** action is executed synchronously by all components

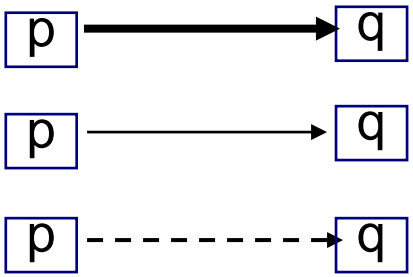
Synchronous Systems – Automata based Model



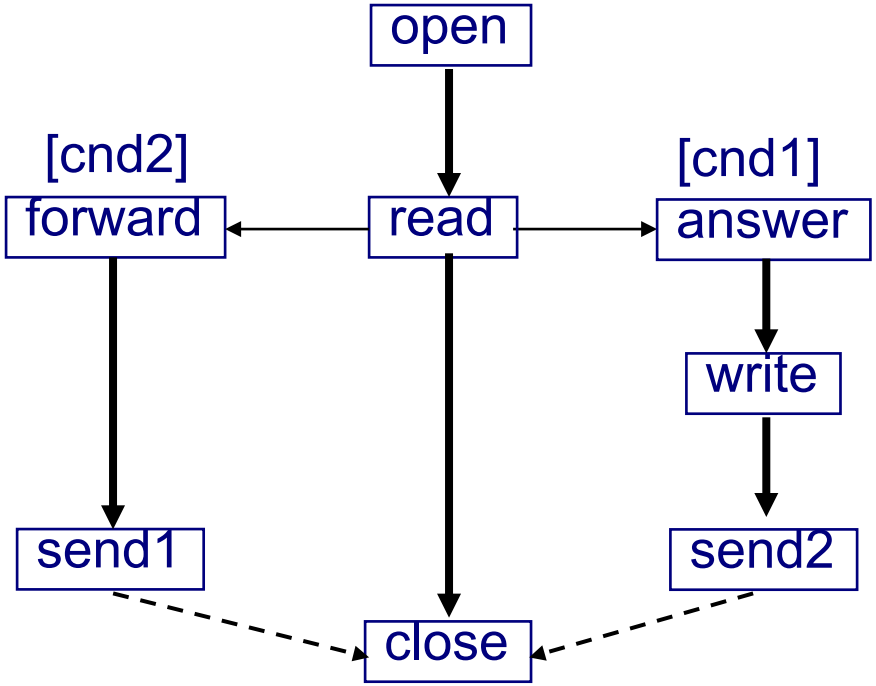
Important issues:

- Strong synchronization may introduce **deadlocks** - How to guarantee that the behaviour is deadlock-free?
- Predictability despite **non-deterministic** behaviour? How to guarantee confluence of computation in a step?

Synchronous Systems – Modal Flow Graphs



strong	q must follow p	pq	possible sequences
weak	q may follow p	p,pq	
conditional	q never precedes p	p,q,pq	



Cyclic treatment of emails

Possible sequences:

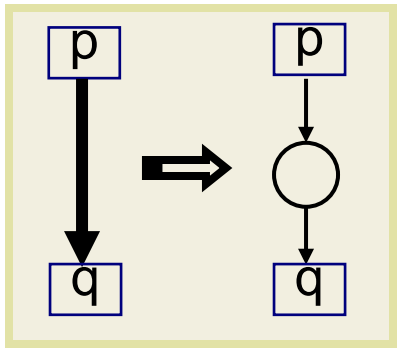
open read close

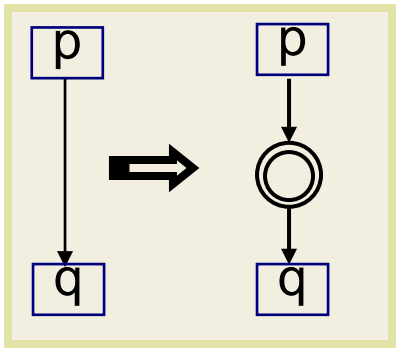
open read forward send1 close

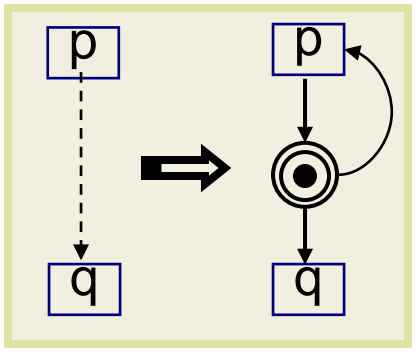
open read answer write send2 close

open read (forward send1 || answer write send2) close

Synchronous Systems – Modal Flow Graphs vs. Petri nets

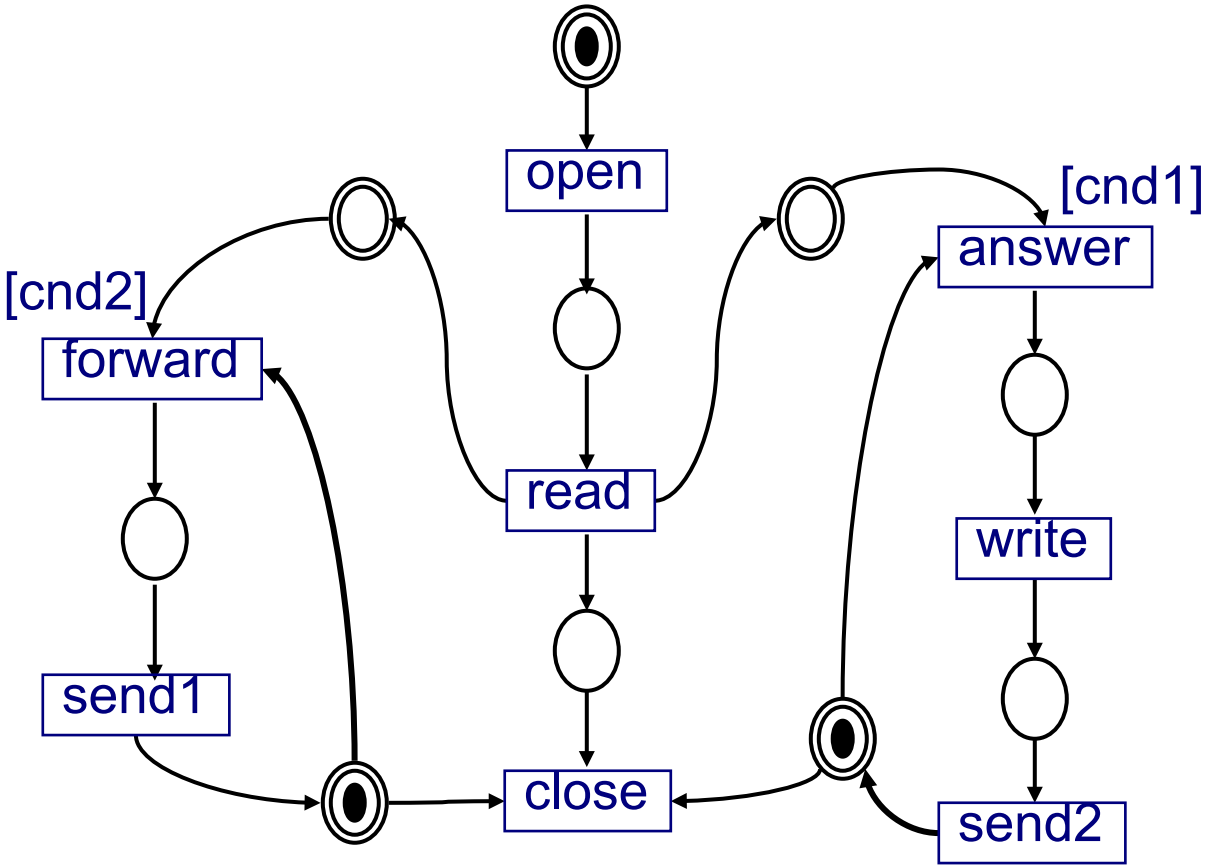






Final place

Initial place

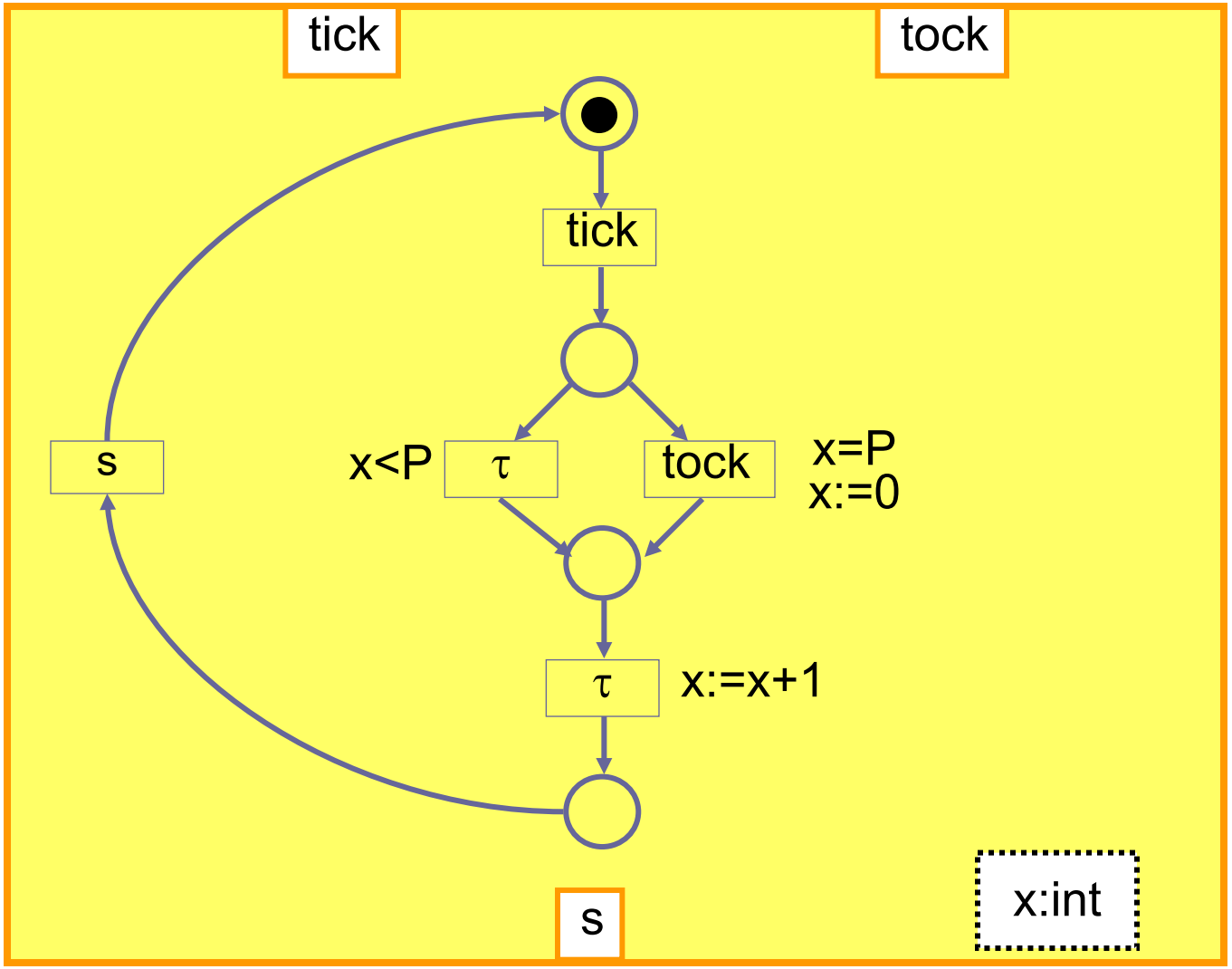


Cyclic treatment of emails

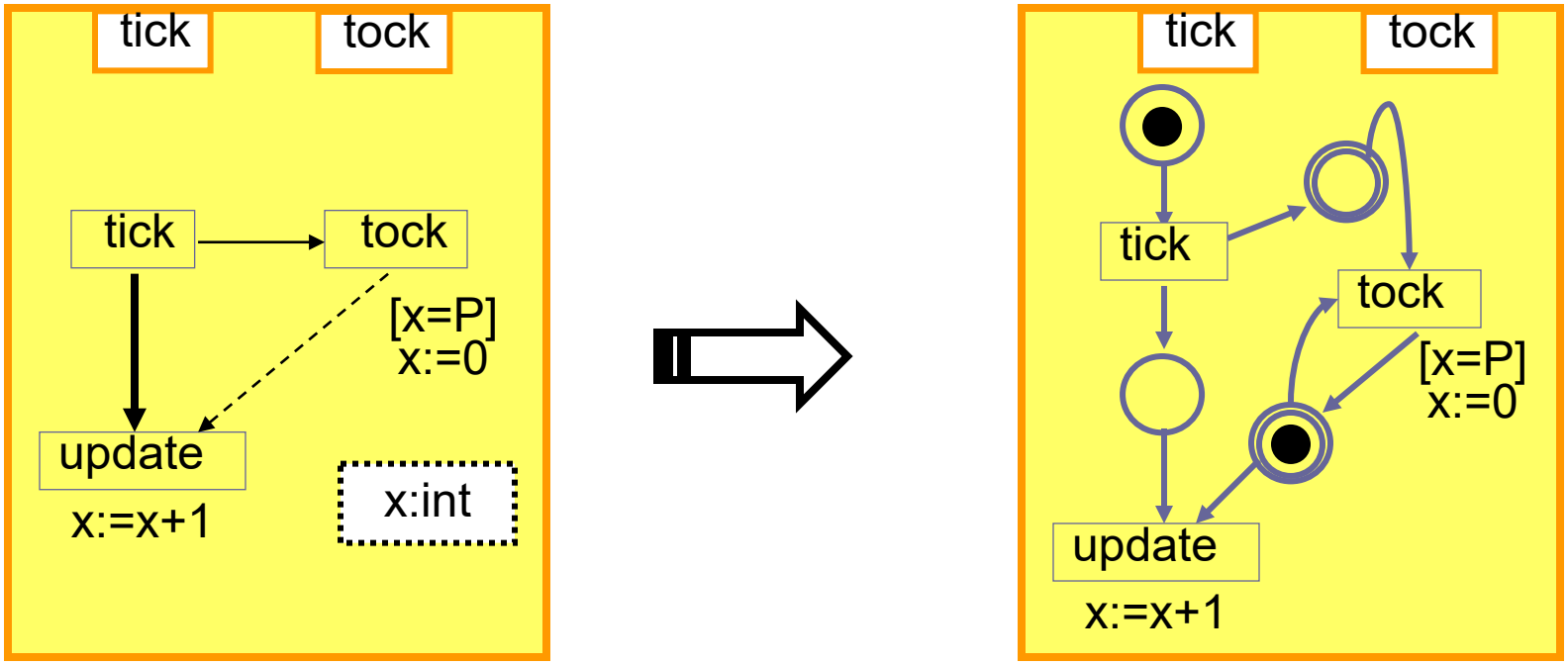
Priorities:

$x \langle y$ if $y \rightarrow x$

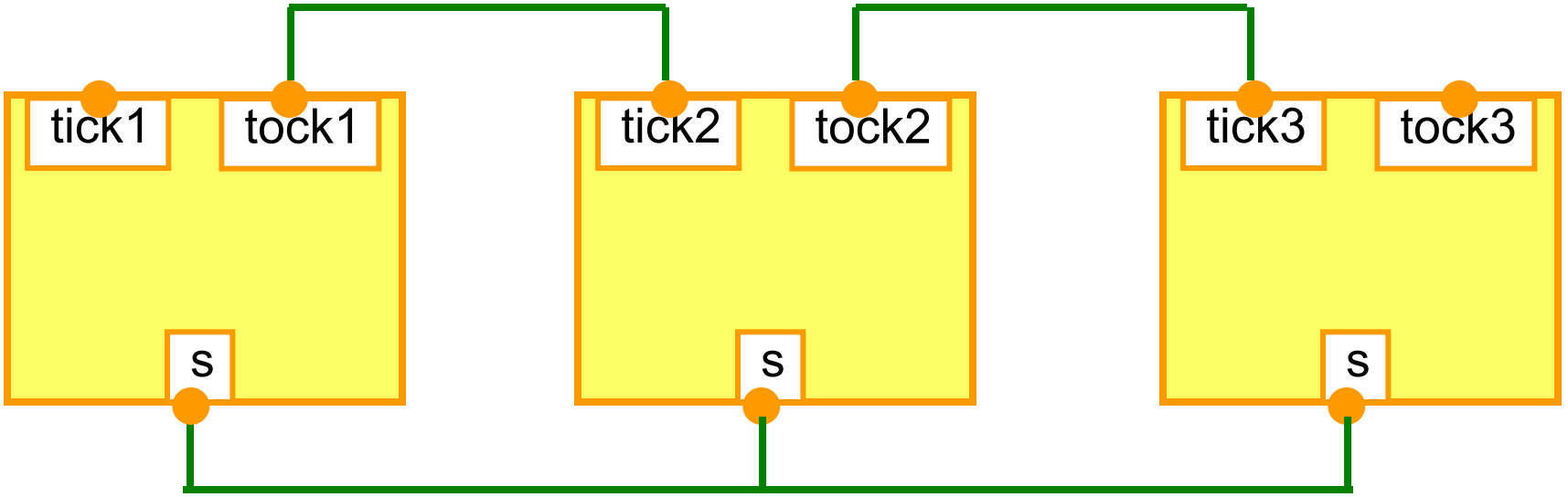
Synchronous Systems –Tick-Tock Component

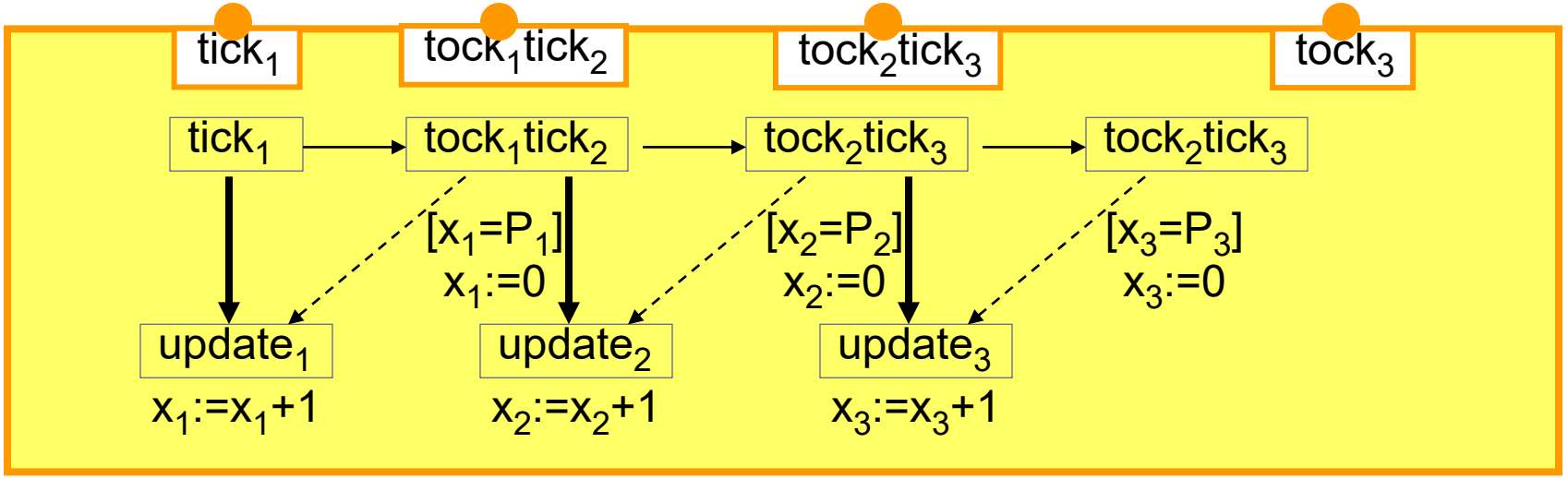


Synchronous Systems –Tick-Tock Component



Synchronous Systems –Tick-Tock Component





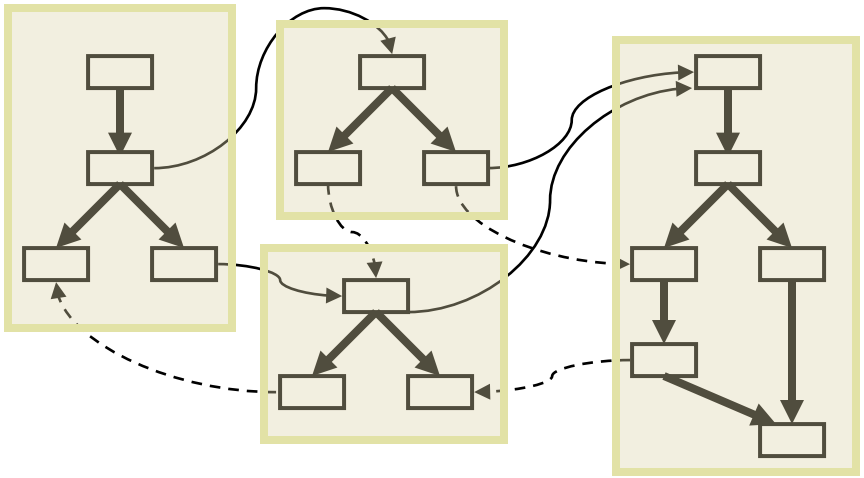
Synchronous Systems – Some Results

A MFG is called well-triggered if,

- each port has a unique minimal strong cause
- each port has exclusively either strong or weak dependencies
- each port with strong dependencies has its guard true

Well-triggered MFG can be decomposed:

- Strong dependencies, define a set of connected graphs
- Weak dependencies, express triggering of roots of these graph
- Conditional dependencies, relate ports of different graphs



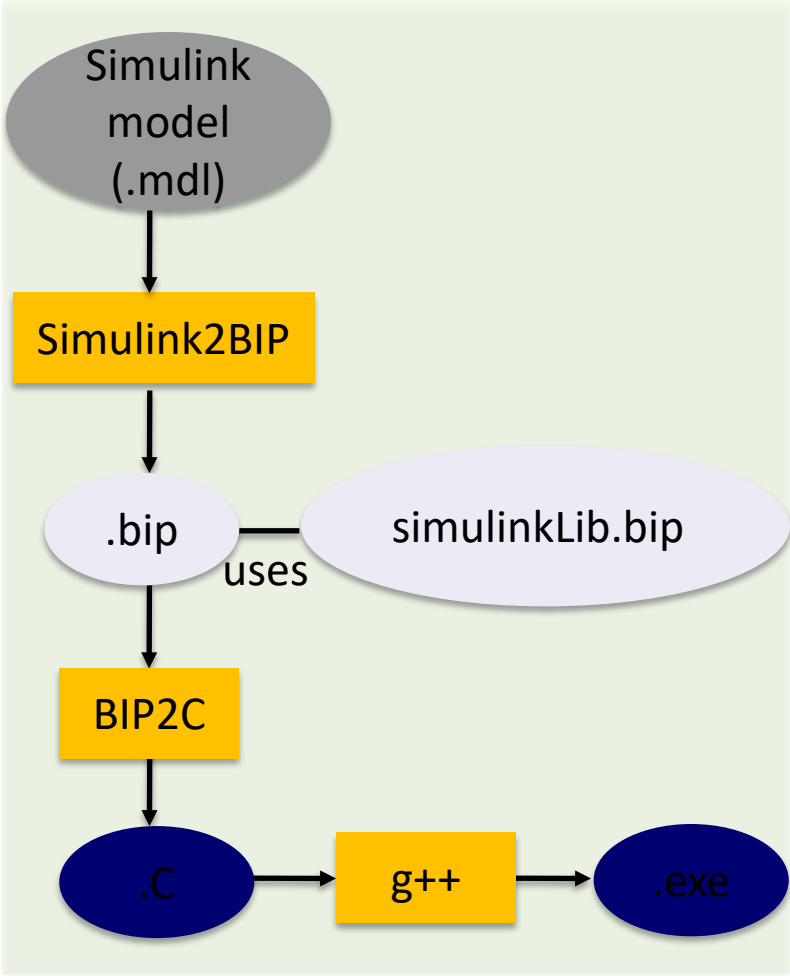
Result:
Well-triggered MFG are deadlock-free and confluent

Synchronous Systems – Translating Simulink into BIP

- Experimental results are obtained for discretized demo examples of MATLAB/Simulink
- For Simulink, executable code is obtained using Real-Time Workshop

Example	*#A*	*#P*	*#T*	*#E*	*n*	*RTW*	*BIP*
Anti-lock breaking	39	2	-	-	10^7	**3.20**	13.51
Steering wheel	120	15	1	-	10^7	**3.42**	16.75
Thermal model	45	3	-	-	10^7	**5.20**	9.62
64-bit counter	365	-	60	-	10^7	53.65	**31.11**
Enabled Subsystem	24	-	-	2	10^7	**3.20**	3.45
Multi Period	14	-	-	1	10^7	4.01	**3.65**

#A: atomic blocks, #P: periodic subsystems, #T: triggered subsystems, #E: enabled subsystems, n: iterations, execution times for Real Time Workshop (RTW) and BIP (secs)



- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

SW Componentization – The Dala Robot

Dala ATRV (iRobot)

- Size (L x W x H): 105cm x 80cm x 100cm
- Weight :120 kg
- Mobility system: 4 driving wheels
- Mobility performances:
  - Max speed: 2 m/s
  - Power supply:Batteries 24V, 1440W
- Payload
  - Sensors:
    - odometry,
    - 12 Sonars,
    - 1 Sick® laser range finder,
    - 2 black and white IEEE1394 cameras,
    - 1 color IEEE1394 panoramic camera.
  - Effectors:
    - Pand and Tilt Unit (Direct Perceptive)
- Processors: 1 Intel Pentium IV
- Communication: IEEE802.11b wireless lan (11Mb/s)
- Operating system: Linux (2.6.10 kernel)

SW Componentization – The Dala Robot SW

GenoM/BIP Toolchain

GenoM

Models

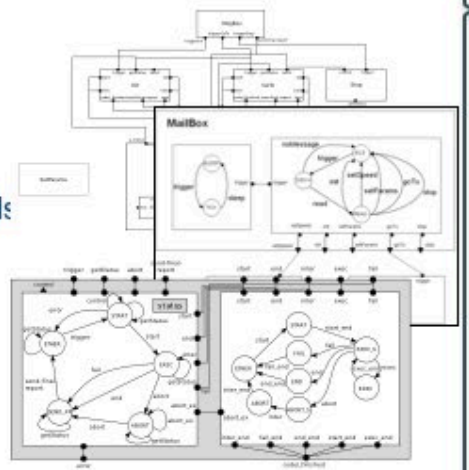
```
module Motion {
  number:  9600;
  SDI:     MOTION_DATA;
}
request SetPos {
  type:    control;
  input:   pos::pos;
  control: controlPos;
  report:  BAD_PARAM;
}
task Move {
  period:   25;
  priority: 15;
}
```

+

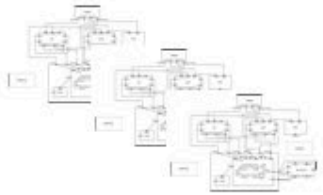
Codels

+

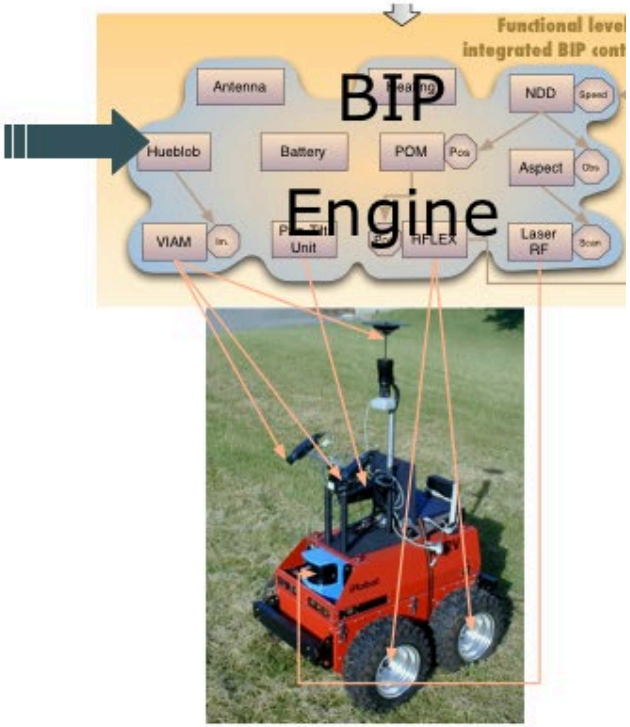
BIP Models



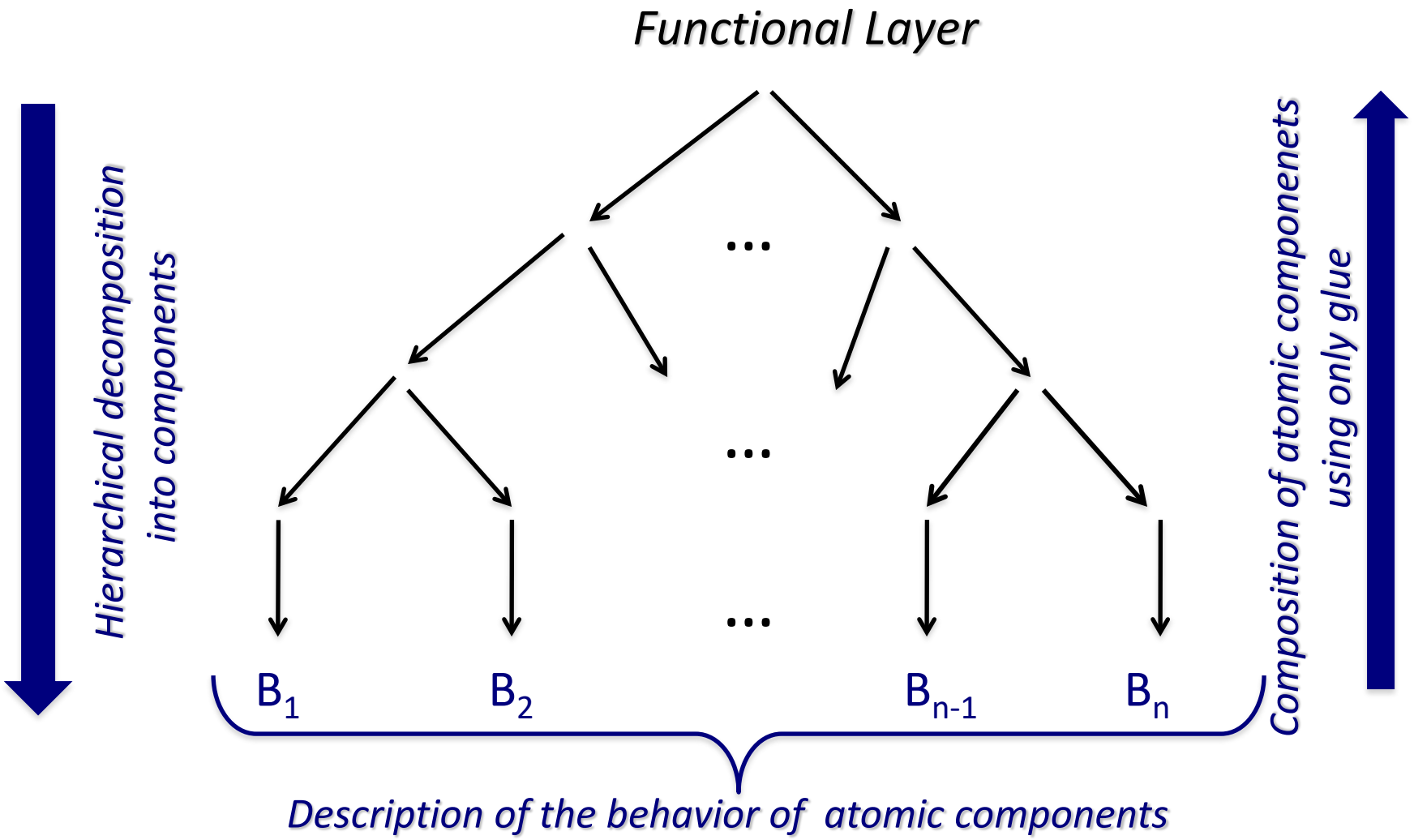
BIP



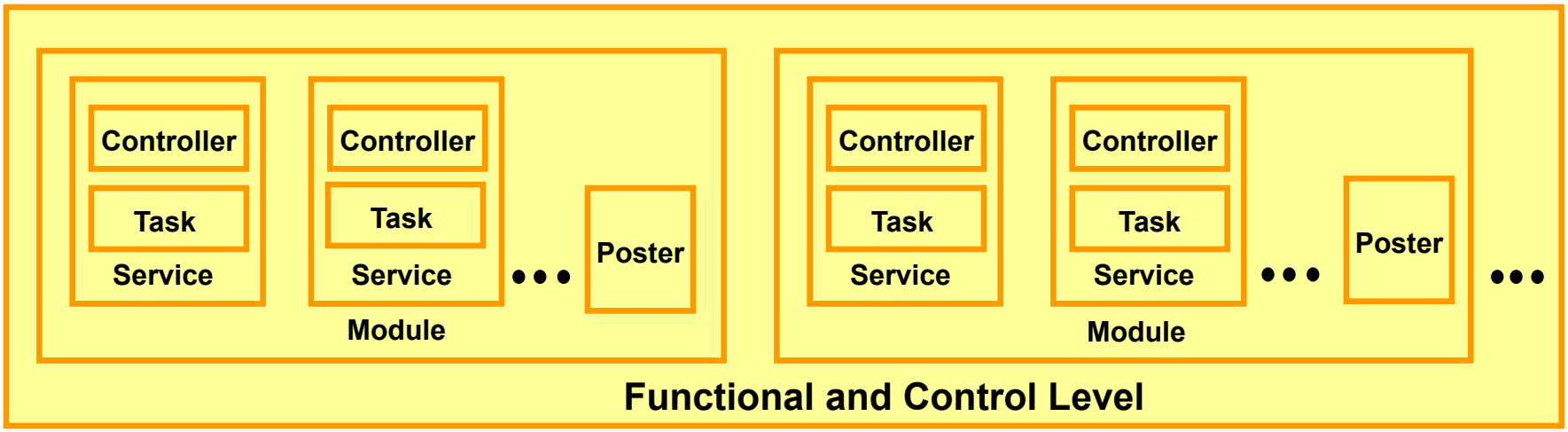
D-Finder



SW Componentization – Methodology



SW Componentization – Methodology



Functional and Control Level ::= Module^{+}

Module ::= Service^{+} . Poster

Service ::= Service Controller . Service Task

Service Controller ::= Event Triggered Controller | Cyclic Controller

Cyclic Controller ::= Event Triggered Controller . Cyclic Trigger

Service Task ::= Timed Task | Untimed Task

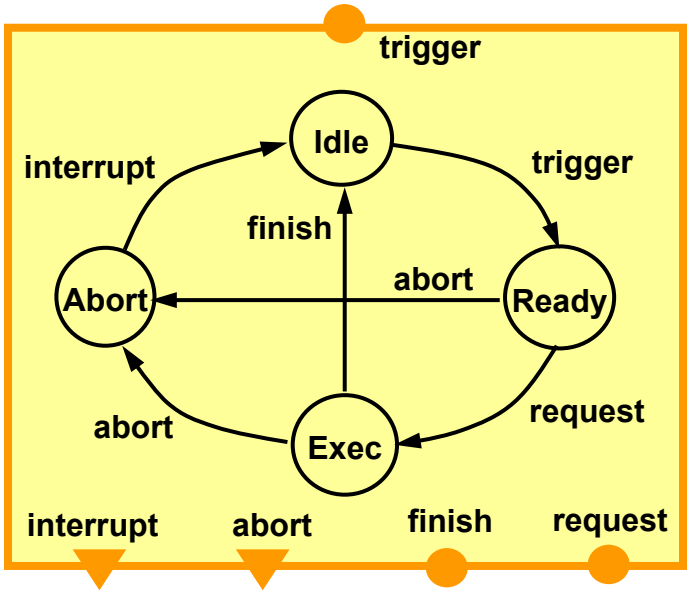
SW componentization – Event Triggered Controller

Idle: the Service is idle

Ready: checks the possibility for starting a new Task of the Service

Exec: execution of the Task of the Service

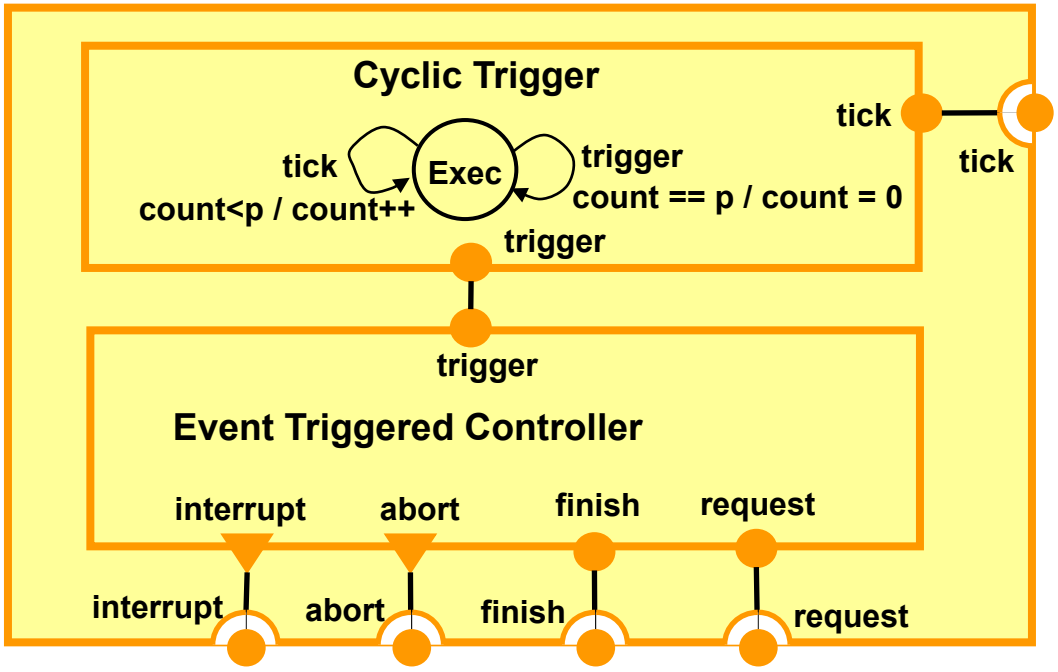
Abort: Service is aborted



SW componentization – Cyclic Controller

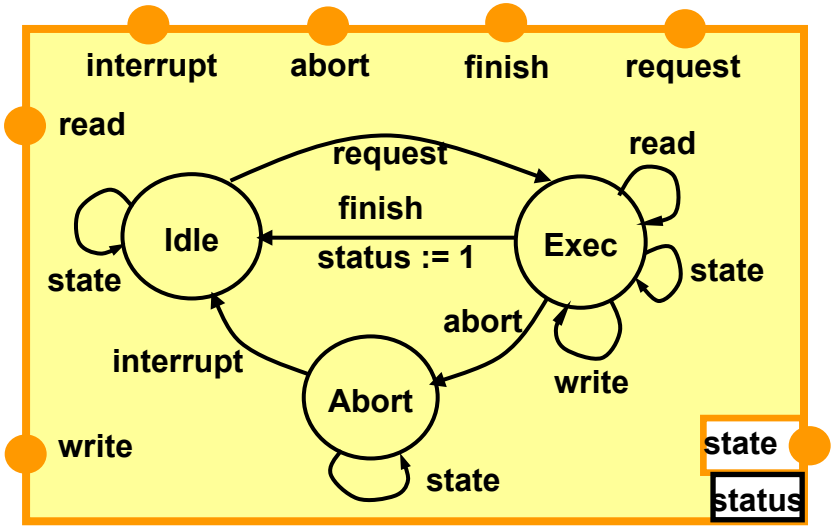
Cyclic Controller ::=
Event Triggered Controller . Cyclic Trigger

The Cyclic Trigger starts the Event Triggered Controller every period p



SW componentization – Untimed Task

Triggered by *request*



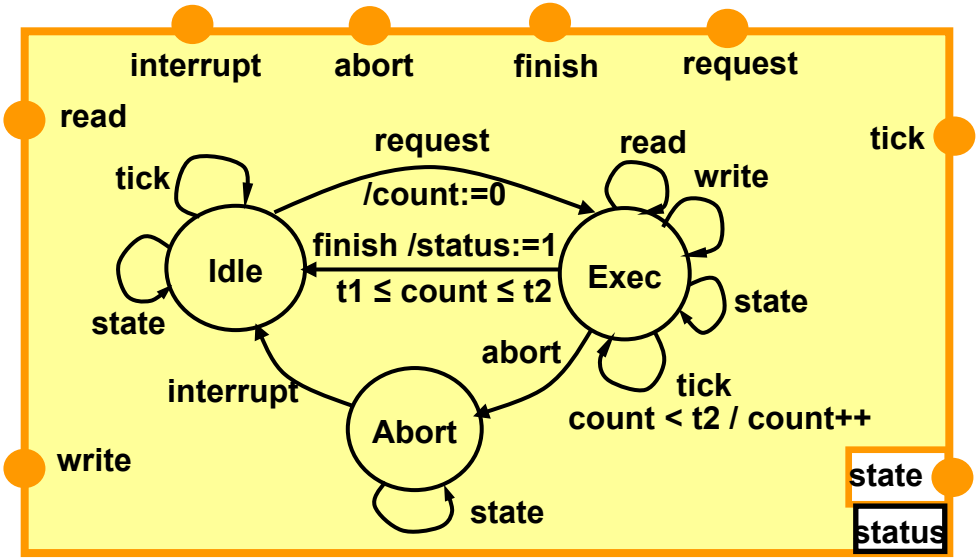
The variable ***status*** specifies the previous state of Task

status == 1 : Task successfully executed

status == 0 : Task aborted

SW componentization – Timed Task

- Obtained from an Untimed Task
- Its execution time is in [t1,t2]

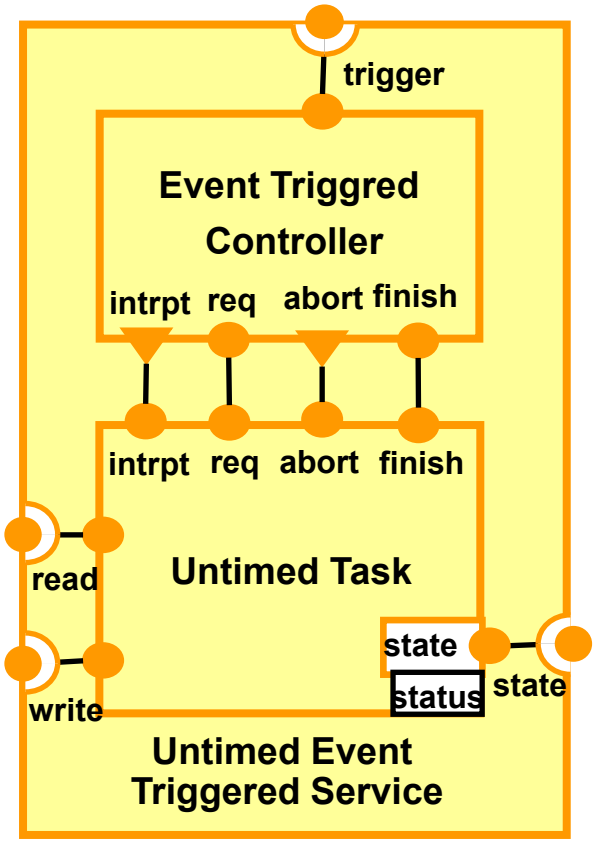


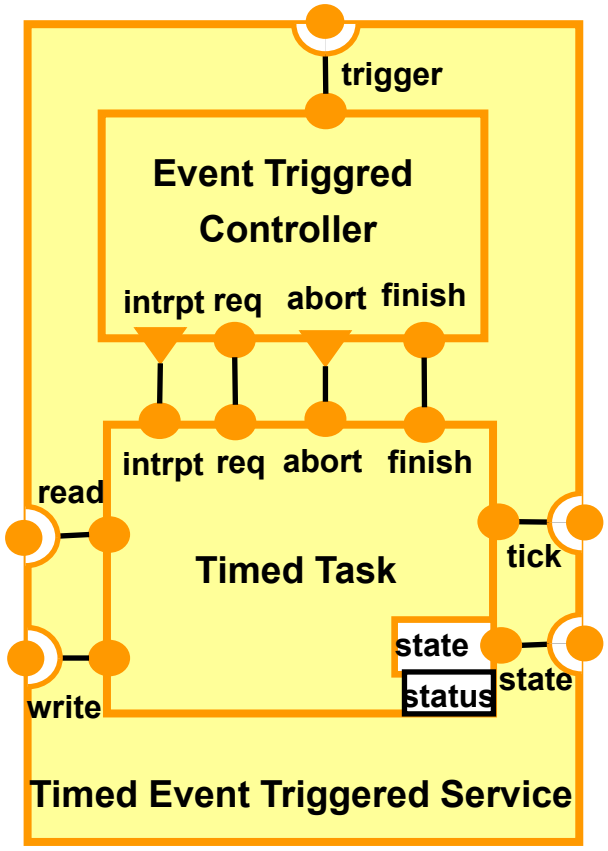
SW componentization – Different types of Services

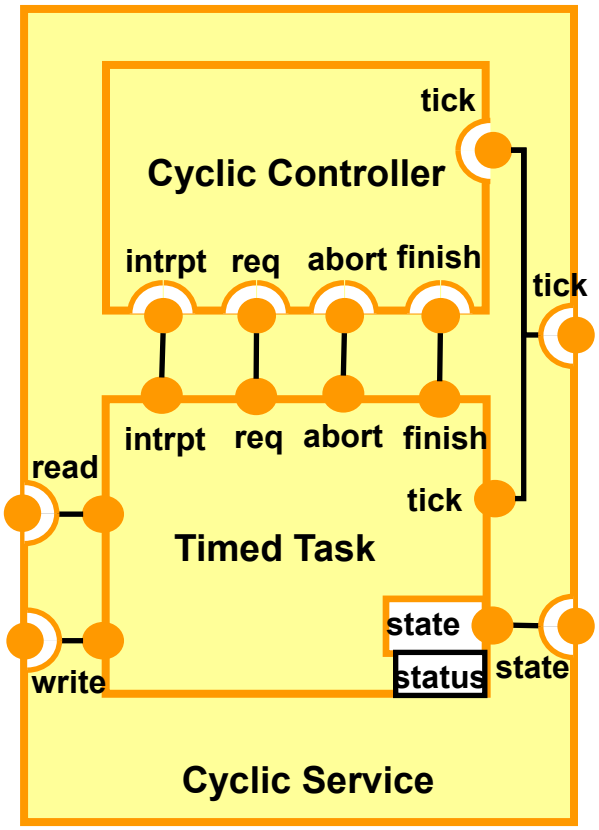
Untimed Event Triggered Service
::= Event Triggered Controller. Untimed Task

Timed Event Triggered Service ::= Event Triggered Controller. Timed Task

Cyclic Service ::= Cyclic Controller . Timed Task

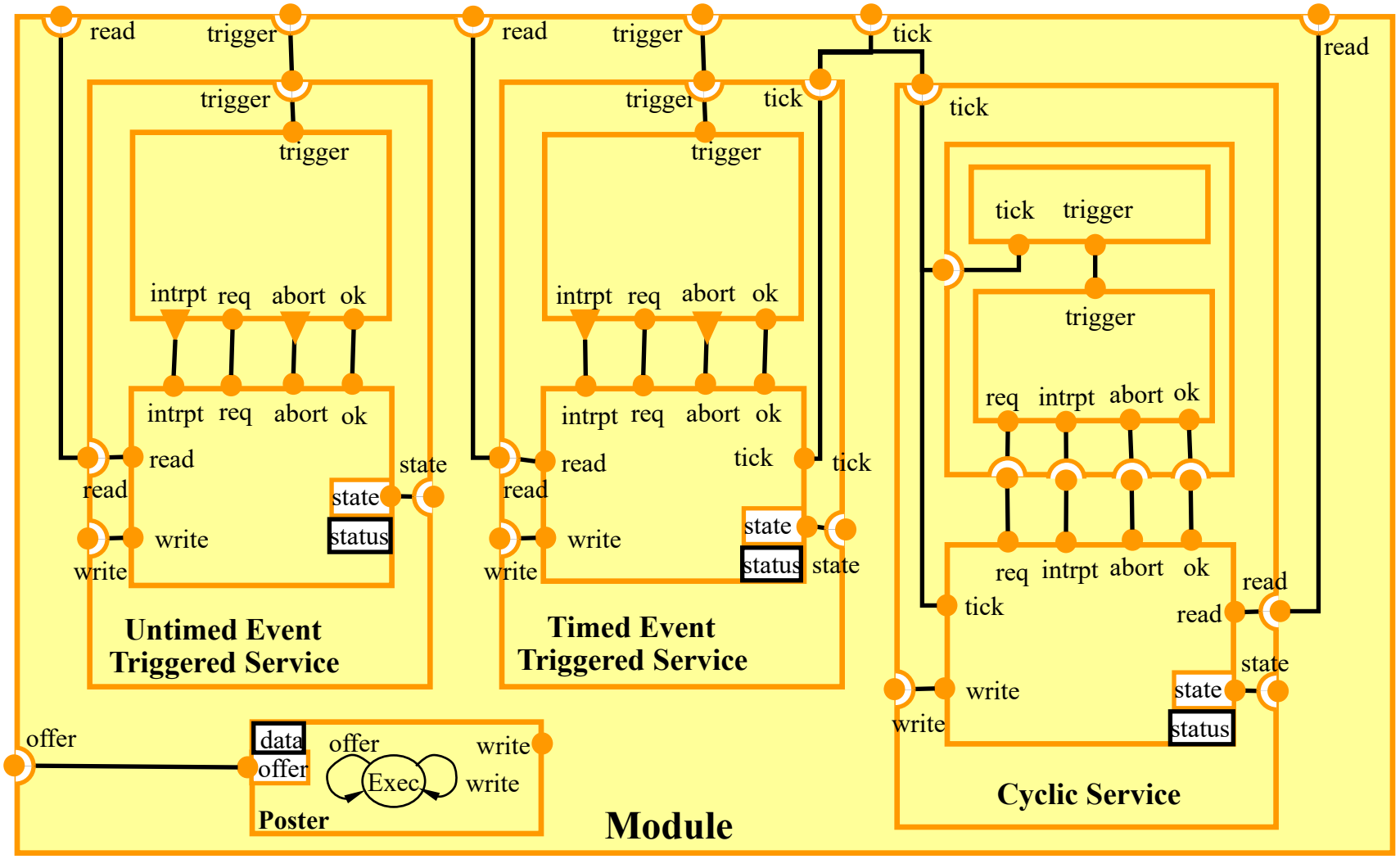






SW componentization – A Module

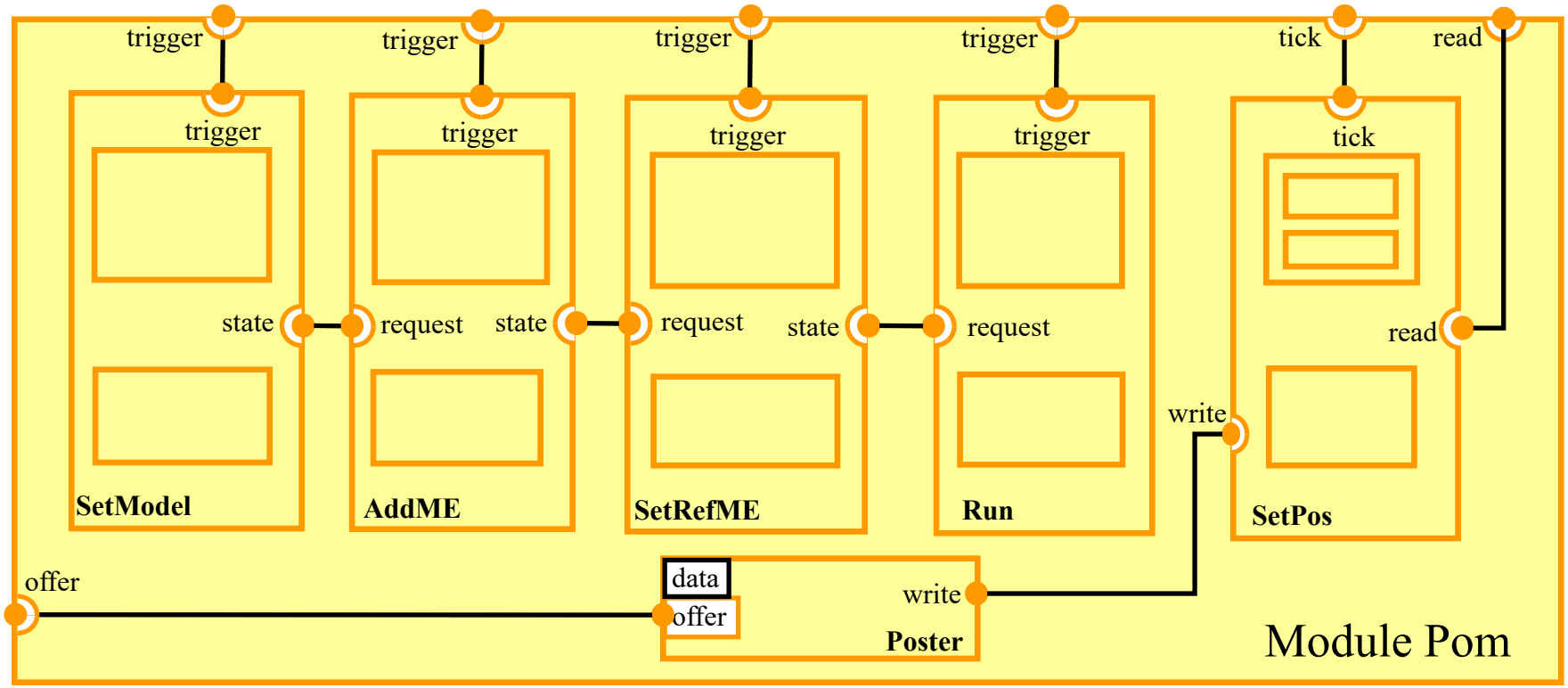
A module composed of 3 services and a poster



SW componentization– Module Pom

Reads and integrates data to provide an estimate of the position of the robot

Pom ::= SetModel. AddME. SetRefME. Run. SetPos. Poster

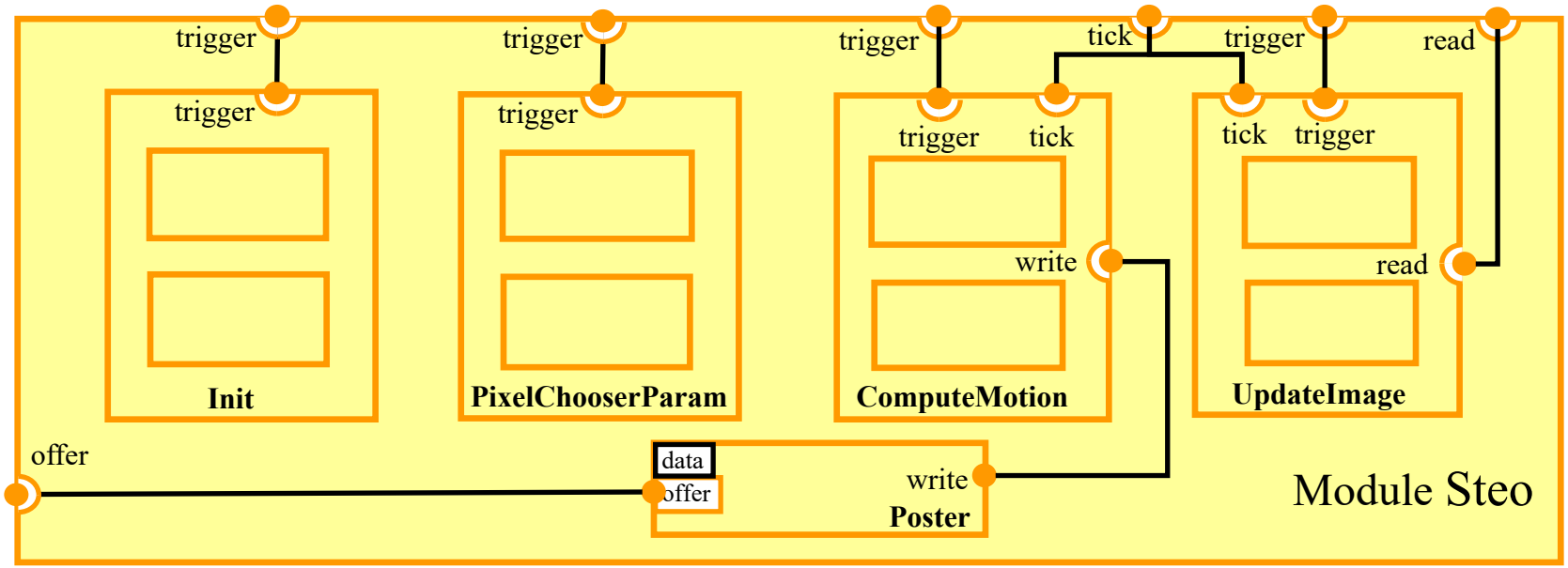


SW componentization – Module Steo

Computes the relative displacement of the robot

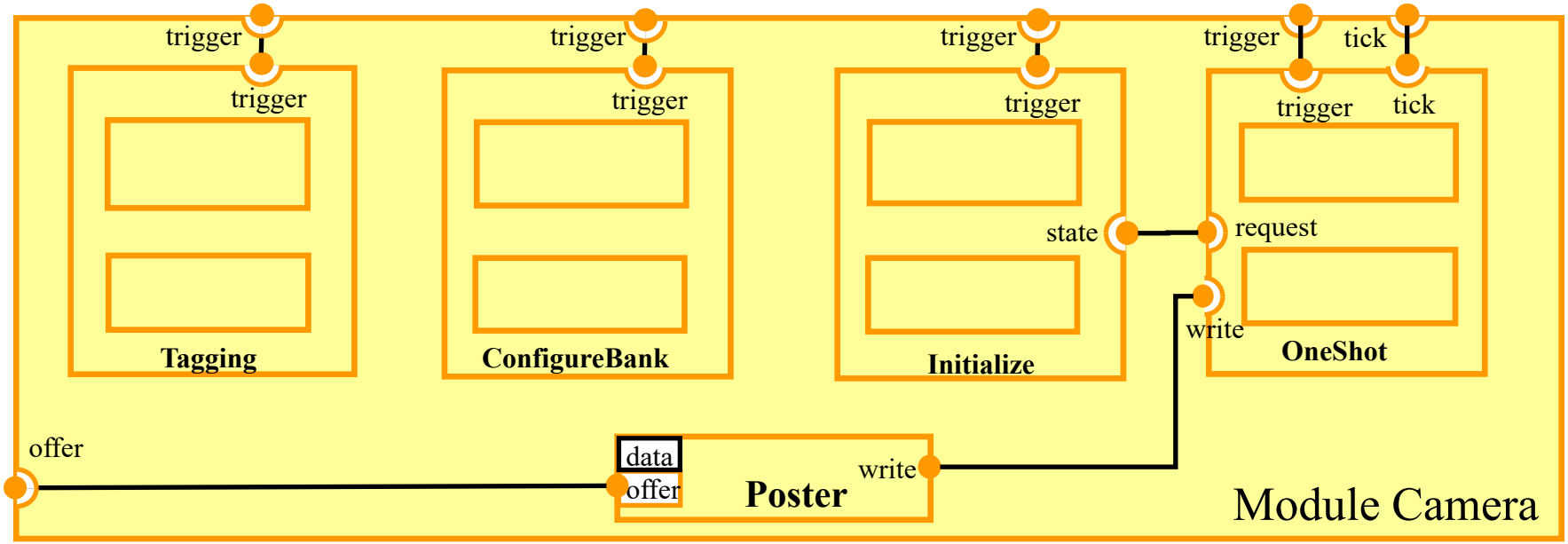
Steo ::=

Init. PixelChooserParam. ComputeMotion. UpdateImage. Poster



SW componentization – Module Camera

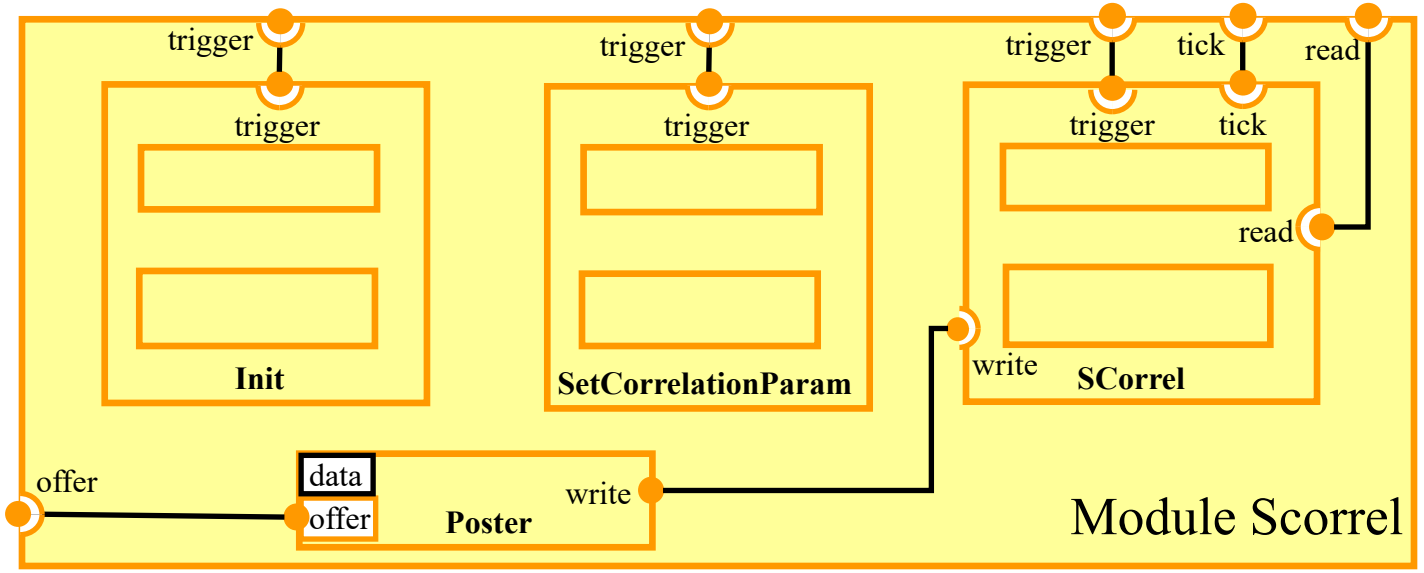
Camera ::= Tagging. ConfigureBank. Initialize. OneShot. Poster



SW componentization – Module Scorrel

Computes the stereo-correlation and produces 3D information about the environment.

Scorrel ::= Init. SetCorrelationParam. SCorrel. Poster



SW componentization – Results

- functional level controller synthesis
  - uses the multithreaded BIP execution engine
  - replaces manually handwritten code
  - the entire functional level is about 500 KLOC
- safety constraints are enforced by construction
  - expressed using interactions and priorities
  - no need to be monitored at runtime
- empirical tests show improved robustness against arbitrary fault injections

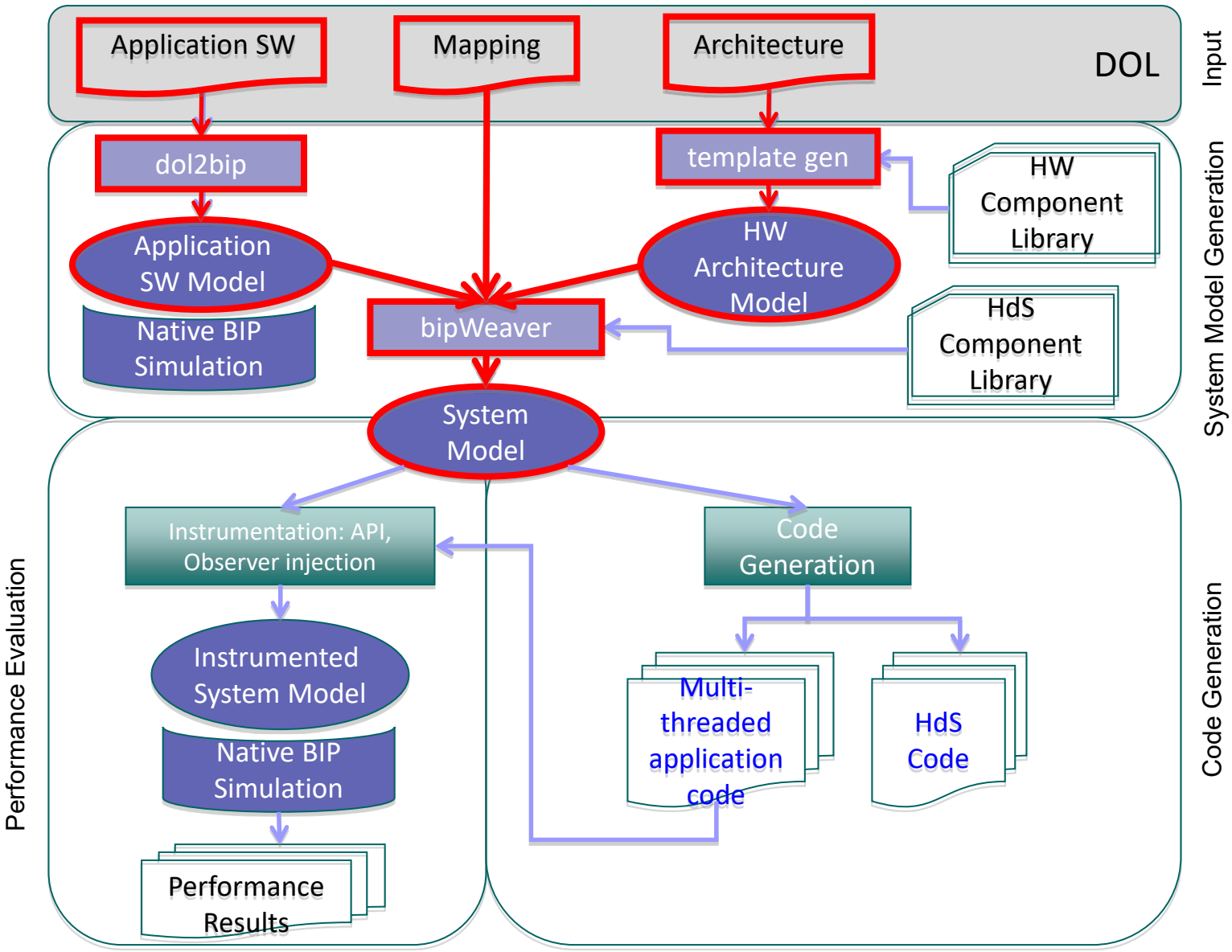
SW componentization – Checking Deadlock-freedom

Modules	Components	Locations	Interactions	States	LOC	Minutes
LaserRF	43	213	202	$2^{20}x3^{29}x34$	4353	1:22
Aspect	29	160	117	$2^{17}x3^{23}$	3029	0:39
NDD	27	152	117	$2^{22}x3^{14}x5$	4013	8:16
RFLEX	56	308	227	$2^{34}x3^{35}x1045$	8244	9:39
Antenna	20	97	73	$2^{12}x3^{9}x13$	1645	0:14
Battery	30	176	138	$2^{22}x3^{17}x5$	3898	0:26
Heating	26	149	116	$2^{17}x3^{14}x145$	2453	0:17
PTU	37	174	151	$2^{19}x3^{22}x35$	8669	0:59
Hueblob	28	187	156	$2^{12}x3^{10}x35$	3170	5:42
VIAM	41	227	231	$2^{10}x3^{6}x665$	5099	4:14
DTM	34	198	201	$2^{28}x3^{20}x95$	4160	13:42
Stereo	33	196	199	$2^{27}x3^{20}x95$	3591	13:20
P3D	50	254	219	$2^{13}x3^{5}x5^{4}x629$	6322	3:51
LaserRF+Aspect+NDD	97	523	438	$2^{58}x3^{66}x85$	11395	40:57
NDD+RFLEX	82	459	344	$2^{56}x3^{49}x5^{2}x209$	12257	73:43

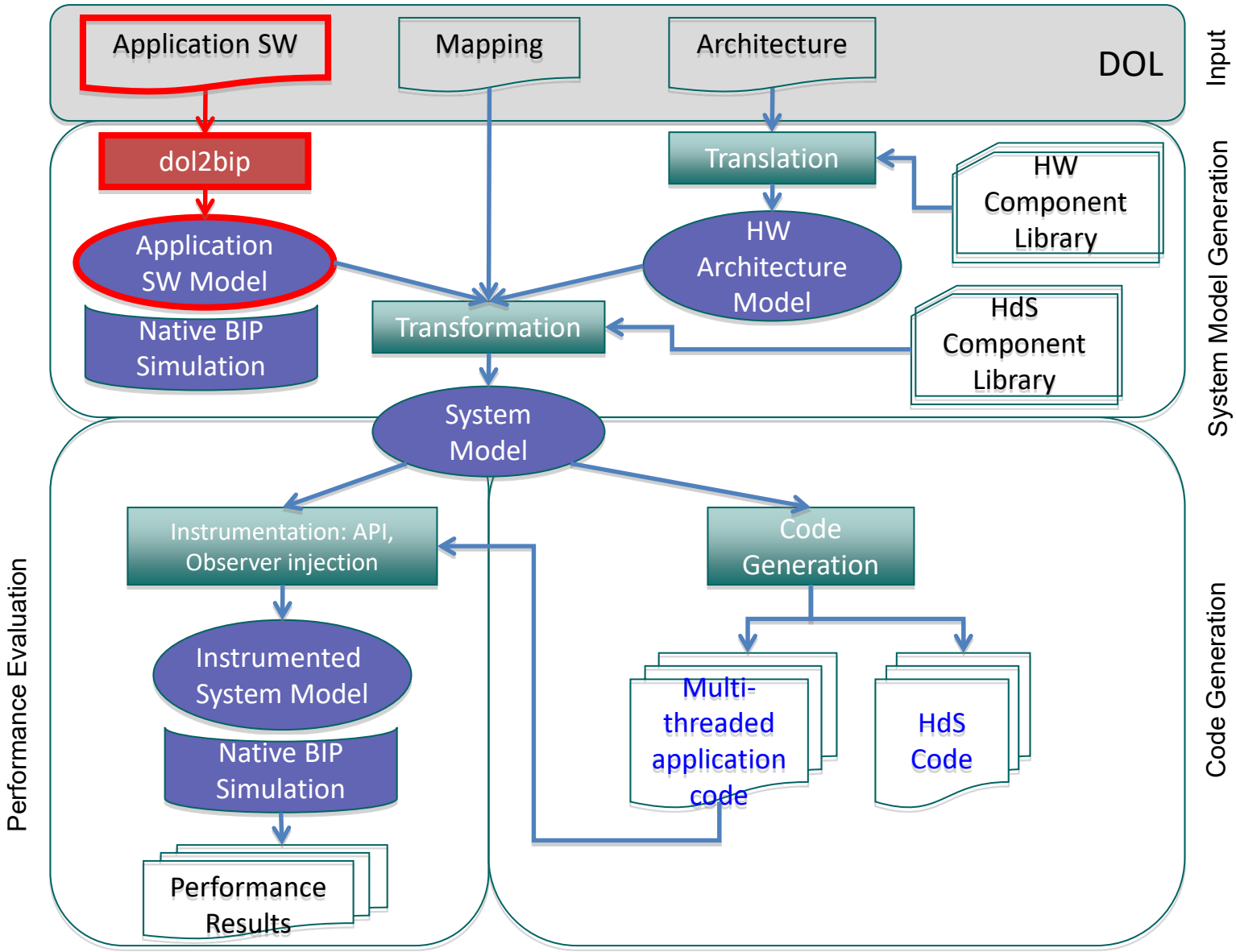
- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

HW-driven refinement – The Design Flow

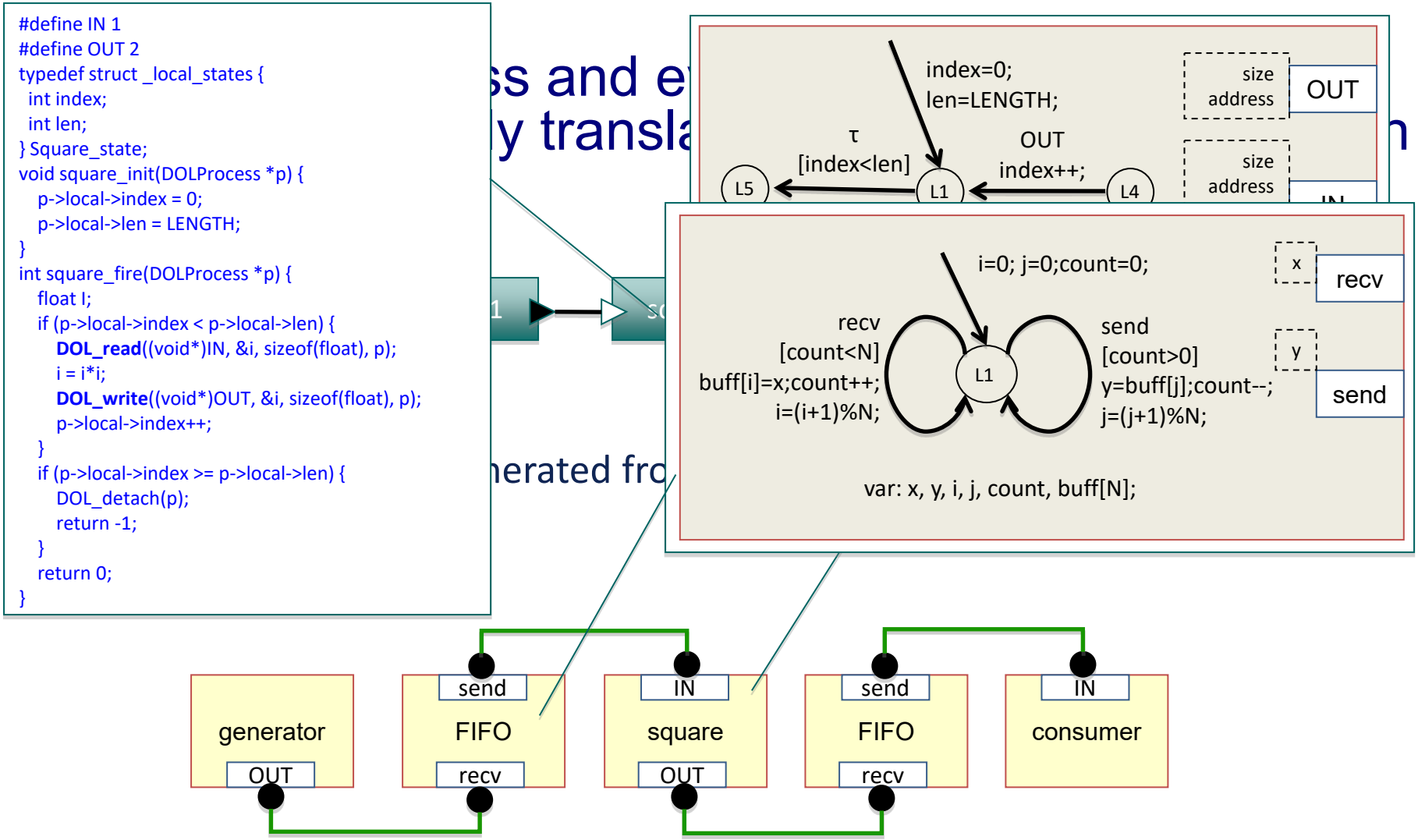


HW-driven refinement – Building the Application SW Model

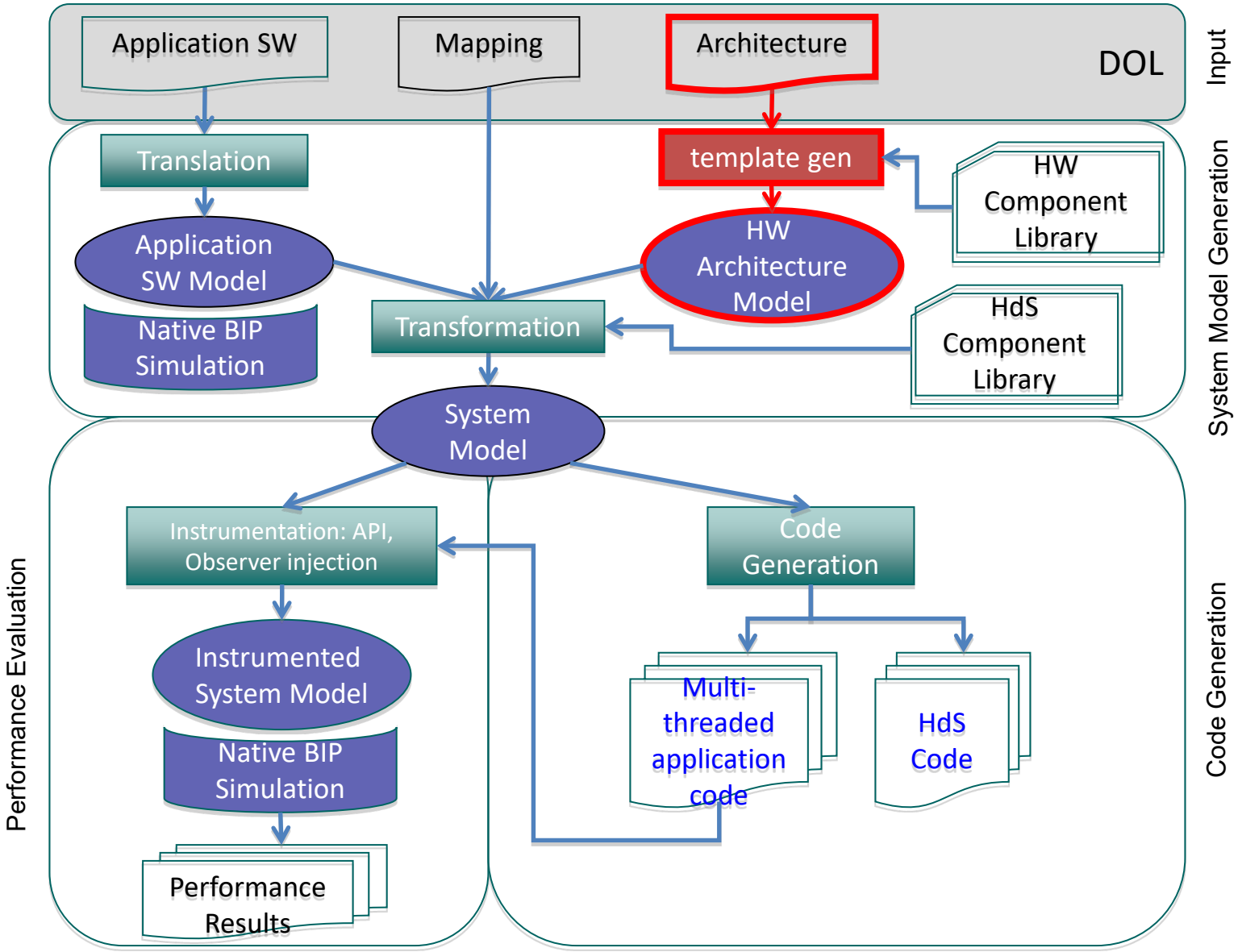


HW-driven refinement – Building the Application SW Model

```
#define IN 1
#define OUT 2
typedef struct _local_states {
  int index;
  int len;
} Square_state;
void square_init(DOLProcess *p) {
    p->local->index = 0;
    p->local->len = LENGTH;
}
int square_fire(DOLProcess *p) {
    float I;
    if (p->local->index < p->local->len) {
        DOL_read((void*)IN, &i, sizeof(float), p);
        i = i*i;
        DOL_write((void*)OUT, &i, sizeof(float), p);
        p->local->index++;
    }
    if (p->local->index >= p->local->len) {
        DOL_detach(p);
        return -1;
    }
    return 0;
}
```



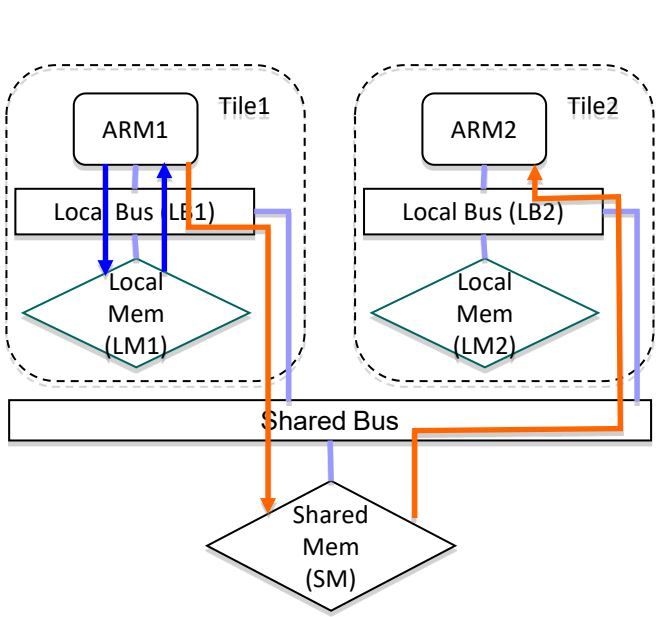
HW-driven refinement – Building the HW Model

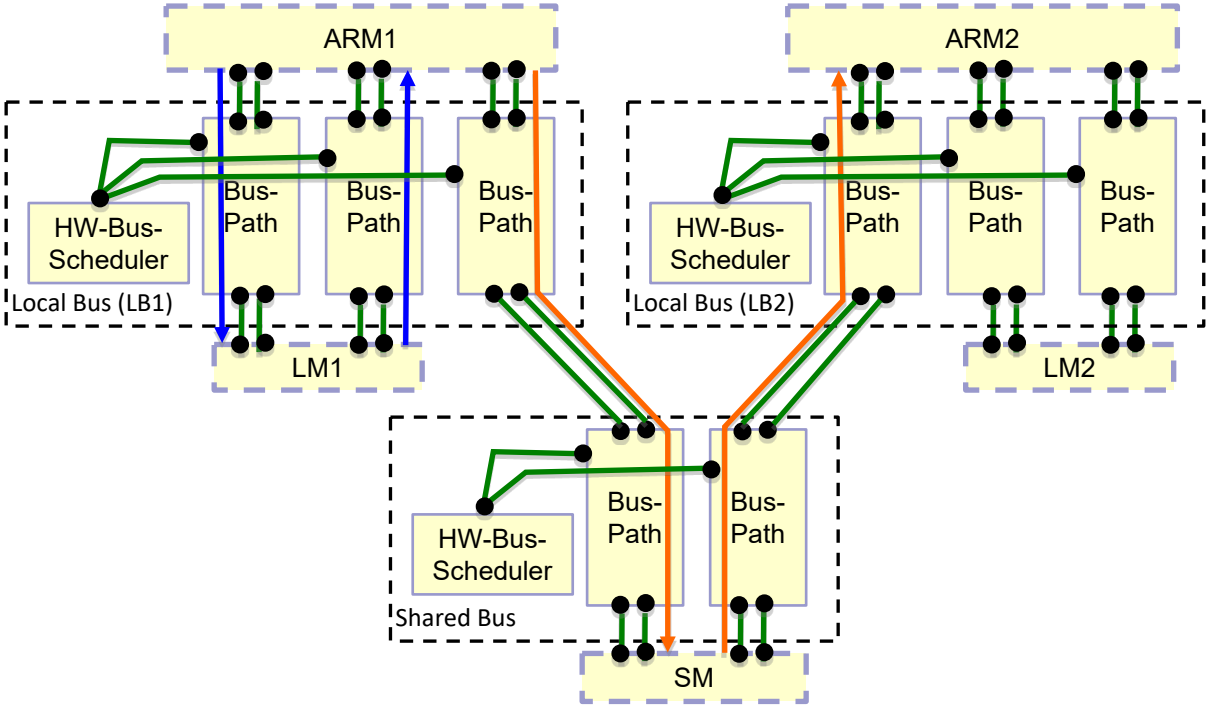


HW-driven refinement – Building the HW Model

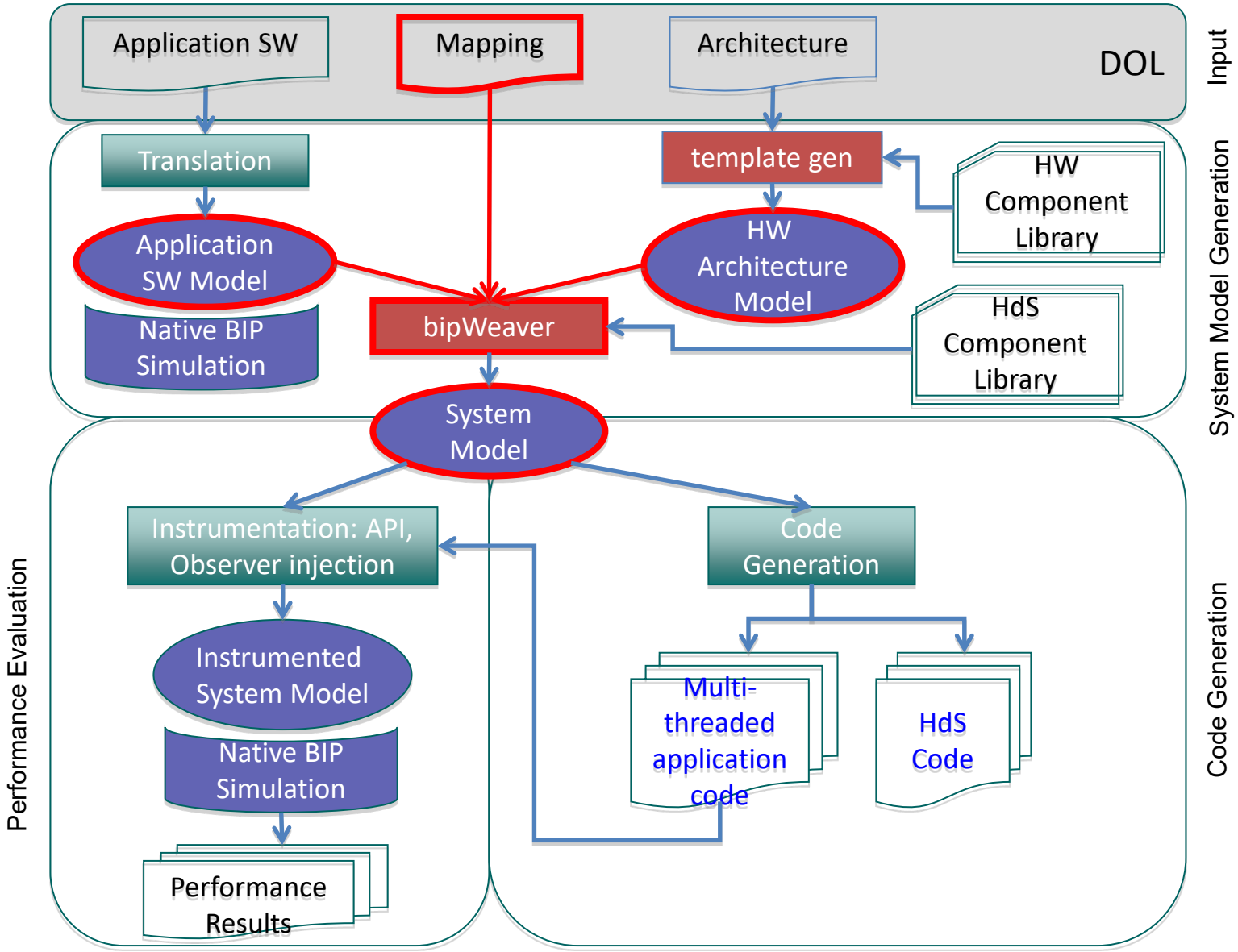
Collection of hw-processor, memory and bus components connected as defined in the architecture

- HW-processor and HW-memory are placeholders
- uses HW component library





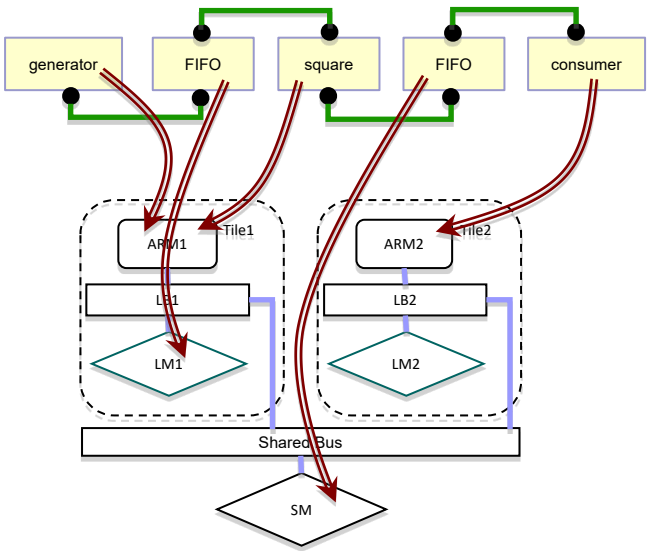
HW-driven refinement – Building the System Model

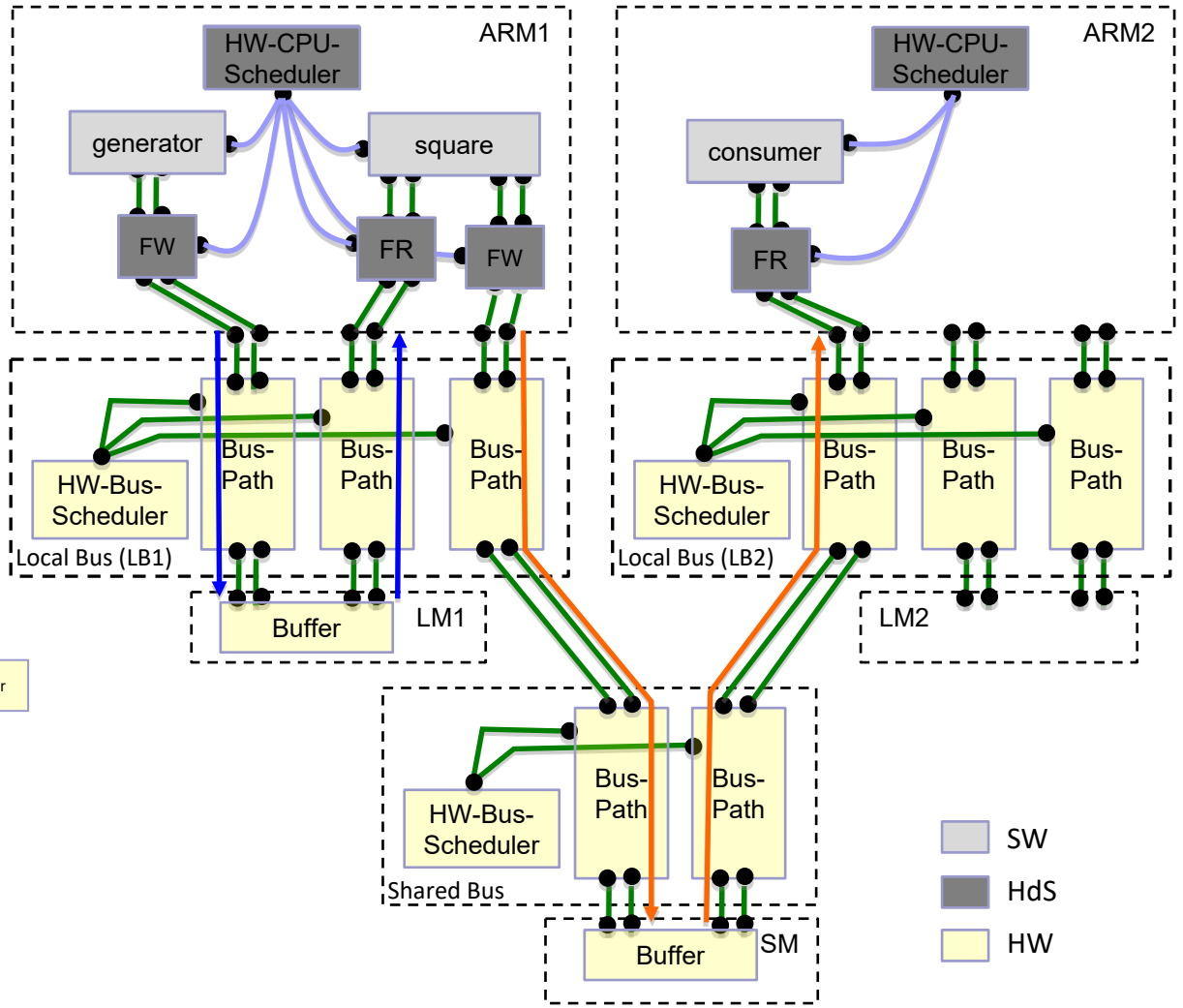


HW-driven refinement – Building the System Model

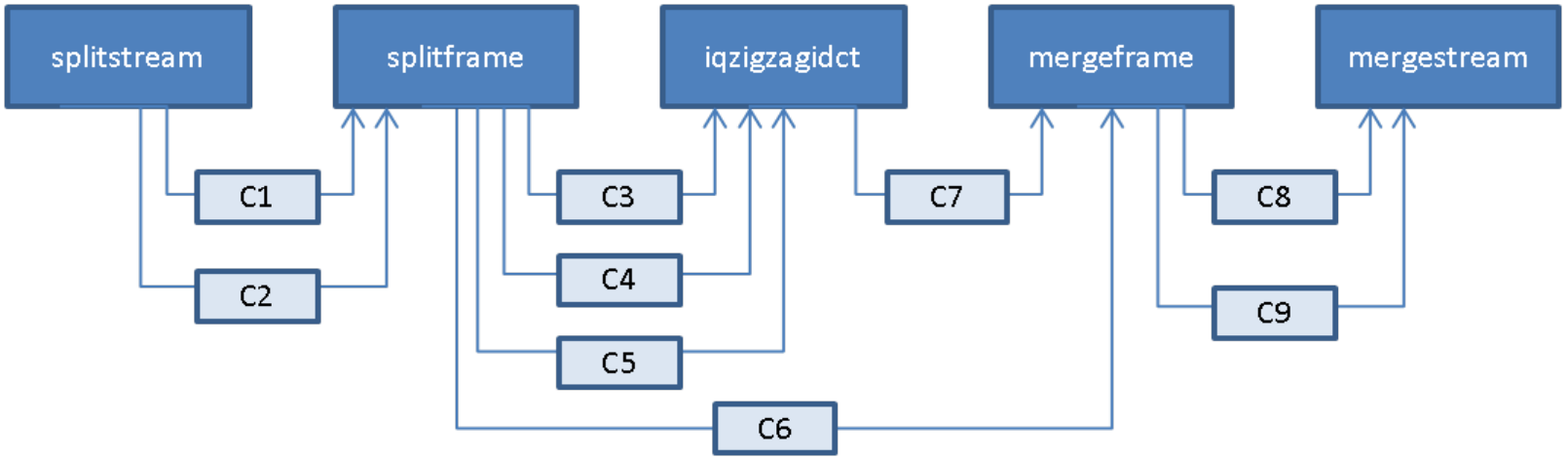
- Transformations defined by the mapping specify how to fill up the HW model
  - fully preserve functional behavior
  - use HdS component library

- Transformation on sw model:
  - Splitting SW-channels
  - Breaking atomic read/write
  - Adding interactions with HW-CPU-Scheduler
  - FIFO buffers mapped to memory





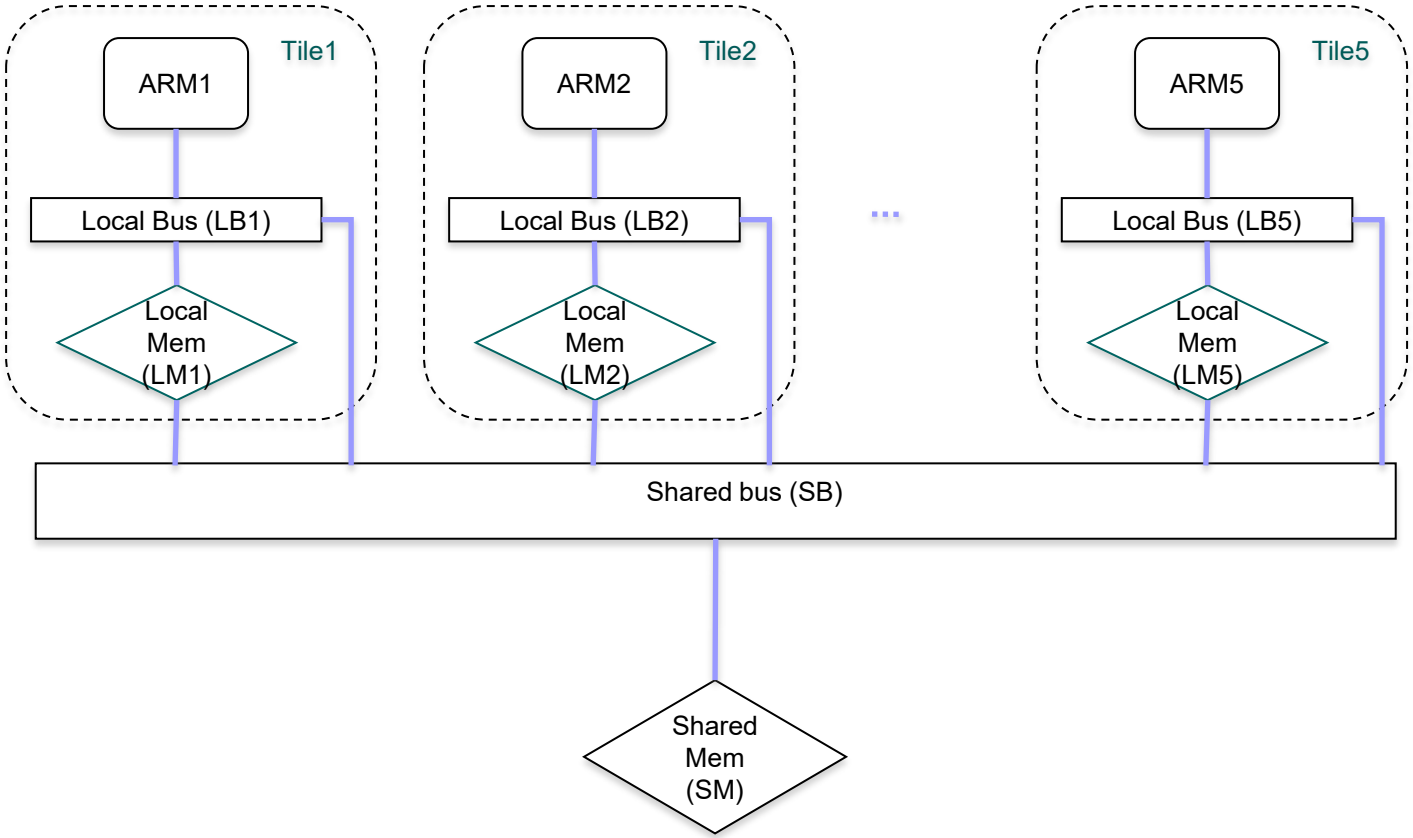
HW-driven refinement – MJPEG decoder



The MJPEG decoder

- reads a sequence of frames and displays the decompressed frames
- is described as a process network with five processes and nine communication channels

HW-driven refinement – MPARM Architecture



A simplified Multi-Processor ARM (MPARM)

- Five identical tiles and a Shared Memory connected via a Shared Bus.
- Tiles contain a CPU connected to its Local Memory via a Local Bus.
- CPU frequency: 200 Mhz.
- Access times: 2 CPU cycles for local memory
  6 CPU cycles for shared memory

HW-driven refinement – MJPEG decoder: Results

Process mapping table

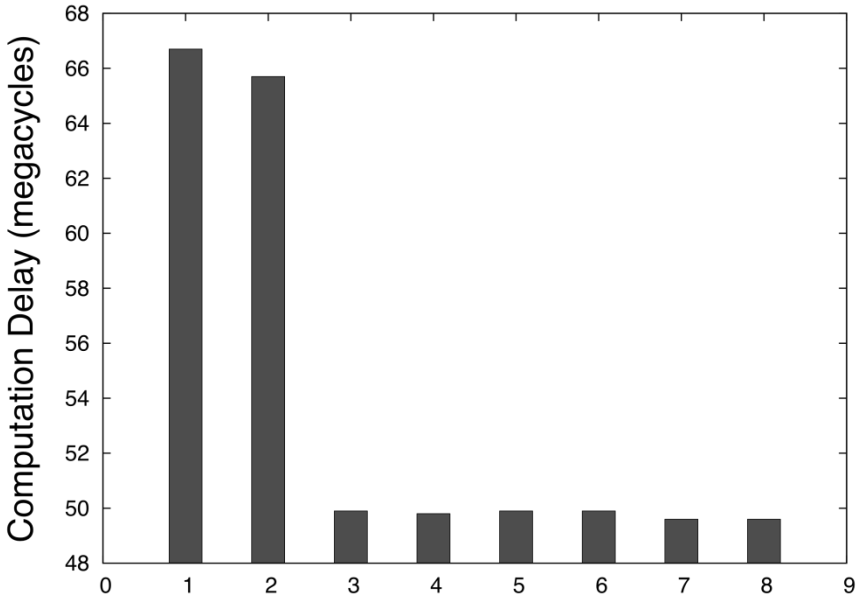
	ARM1	ARM2	ARM3	ARM 4	ARM5
Mapping1	all				
Mapping2	SS, SF, IQ	MF, MS			
Mapping3	SS, SF	IQ,MF,MS			
Mapping4	SS, SF	IQ	MF, MS		
Mapping5	SS, MS	SF	IQ	MF	
Mapping6	SS	SF	IQ	MF	MS
Mapping7	SS, SF	IQ	MF, MS		
Mapping8	SS	SF	IQ	MF	MS

SW-Channel mapping table

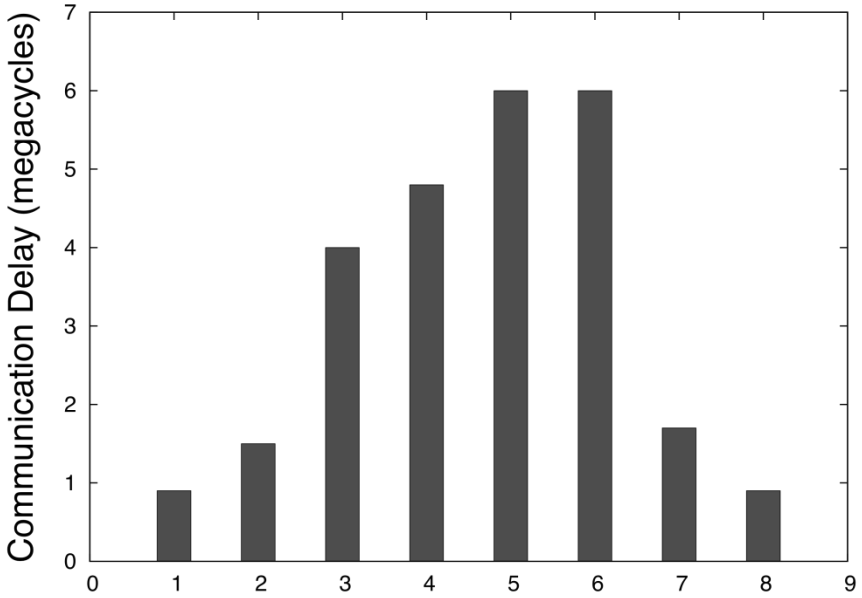
	Shared	LM1	LM2	LM3	LM4
Mapping1		all			
Mapping2	C6, C7	C1, C2, C3, C4, C5	C8, C9		
Mapping3	C3, C4, C5, C6	C1, C2	C7, C8, C9		
Mapping4	C3, C4, C5, C6, C7	C1, C2		C8, C9	
Mapping5	all				
Mapping6	all				
Mapping7	C6, C7	C1, C2, C3, C4, C5		C8, C9	
Mapping8		C1, C2	C3, C4, C5, C6	C7	C8, C9

HW-driven refinement – MJPEG decoder: Results

Computation delay



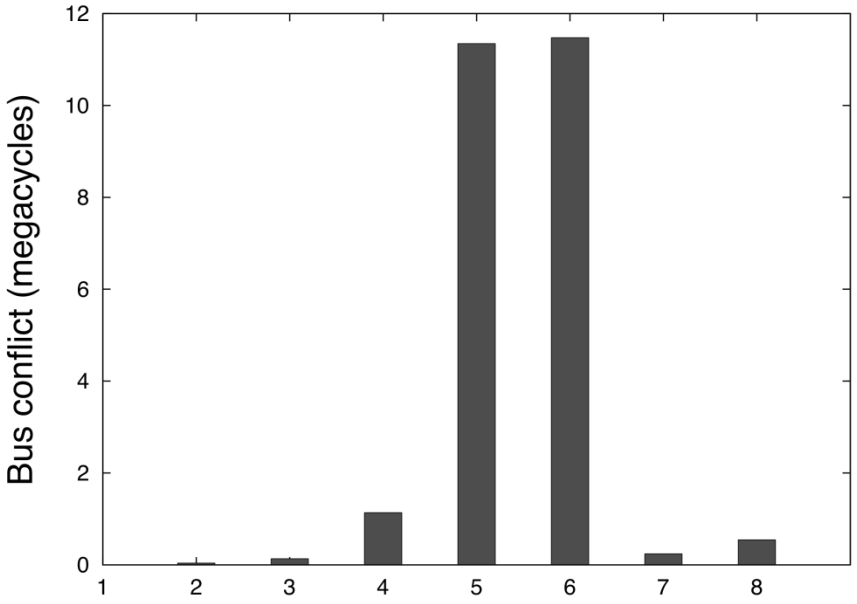
Communication delay

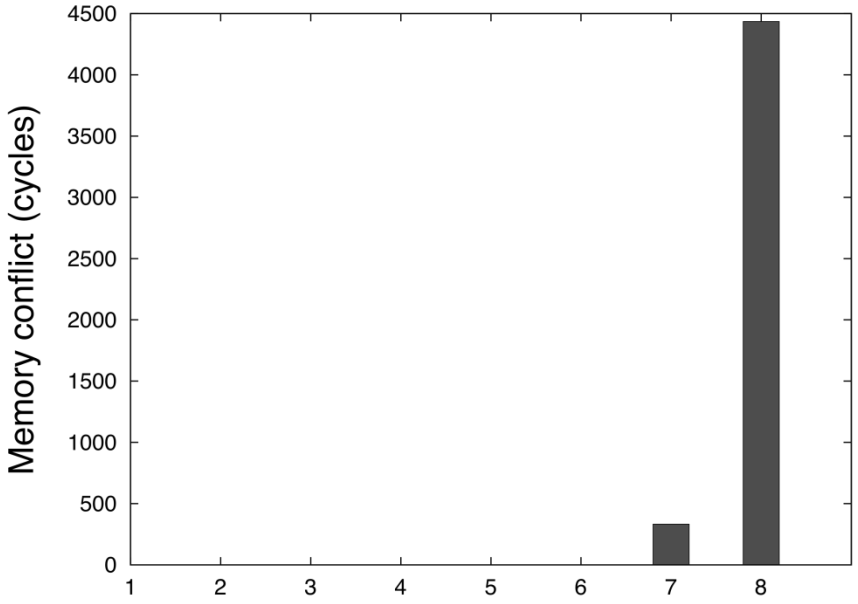


- Mapping (1) gives the worst computation time as all processes are mapped to a single processor.
- The communication overhead is reduced if we distribute sw-channels to the local memories of the processors.

HW-driven refinement – MJPEG decoder: Results

Delay (waiting time) due to bus and memory conflict





As more channels are mapped to the local memory, the shared bus contention is reduced. However, this might increase the local memory contention, as is evident for mapping (8).

- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Distributed Implementation – From BIP to S/R-BIP

BIP is based on:

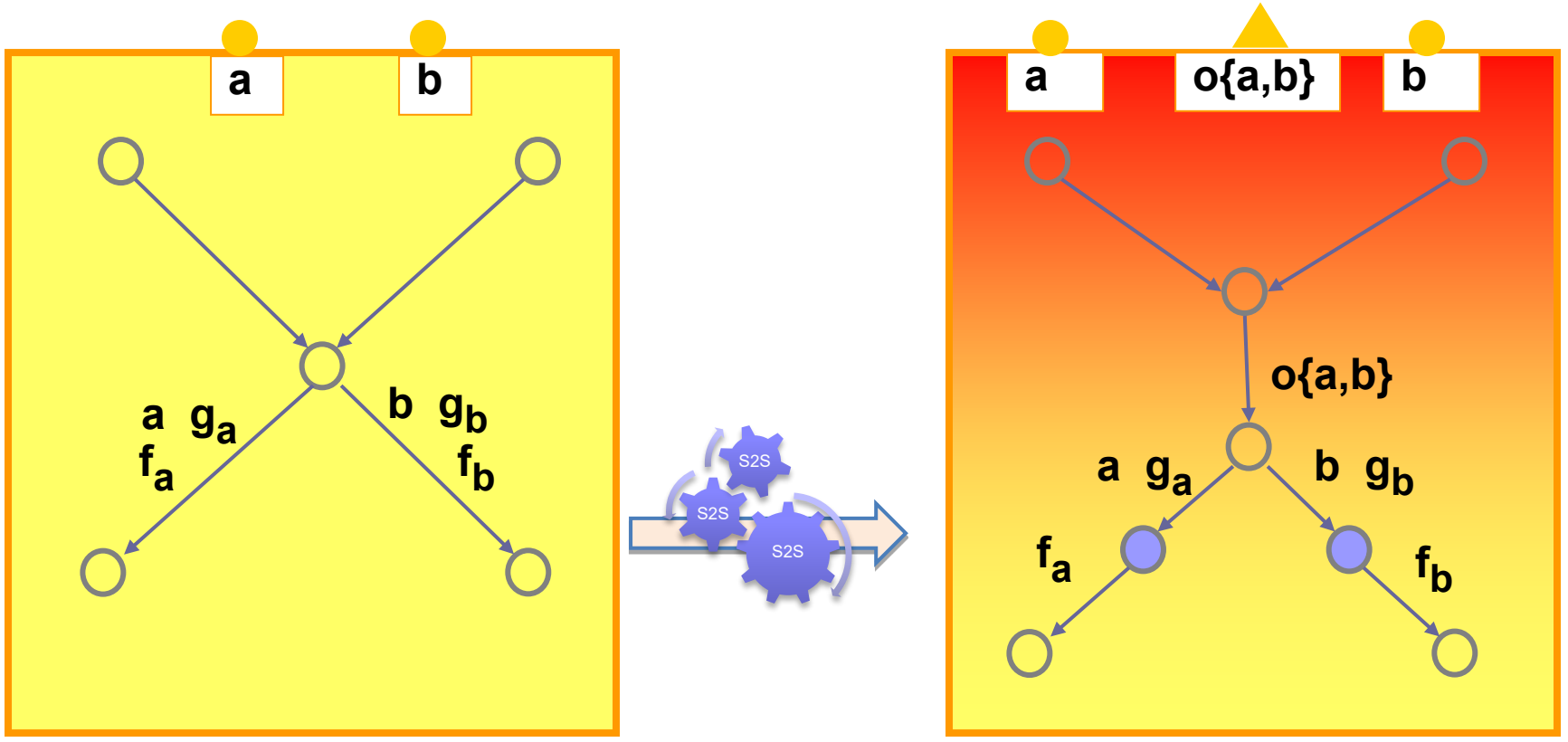
- Global state semantics, defined by operational semantics rules, implemented by the BIP Engine
- Atomic multiparty interactions, e.g. by rendezvous or broadcast

Correct-by-construction translation of BIP models into observationally equivalent S/R-BIP models

- Point to point communication by asynchronous message passing
- No global state - Atomicity of transitions is broken by separating interaction from internal computation
- The BIP Engine is replaced by a set of Engines executing subsets of interactions
- Distributed coordination is orchestrated by an architecture

Distributed Implementation – From BIP to S/R-BIP

Transformation of atomic components



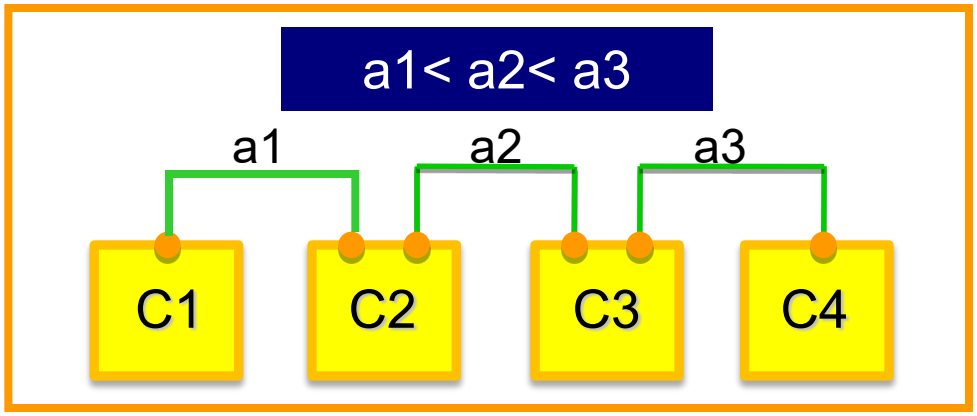
Global state model

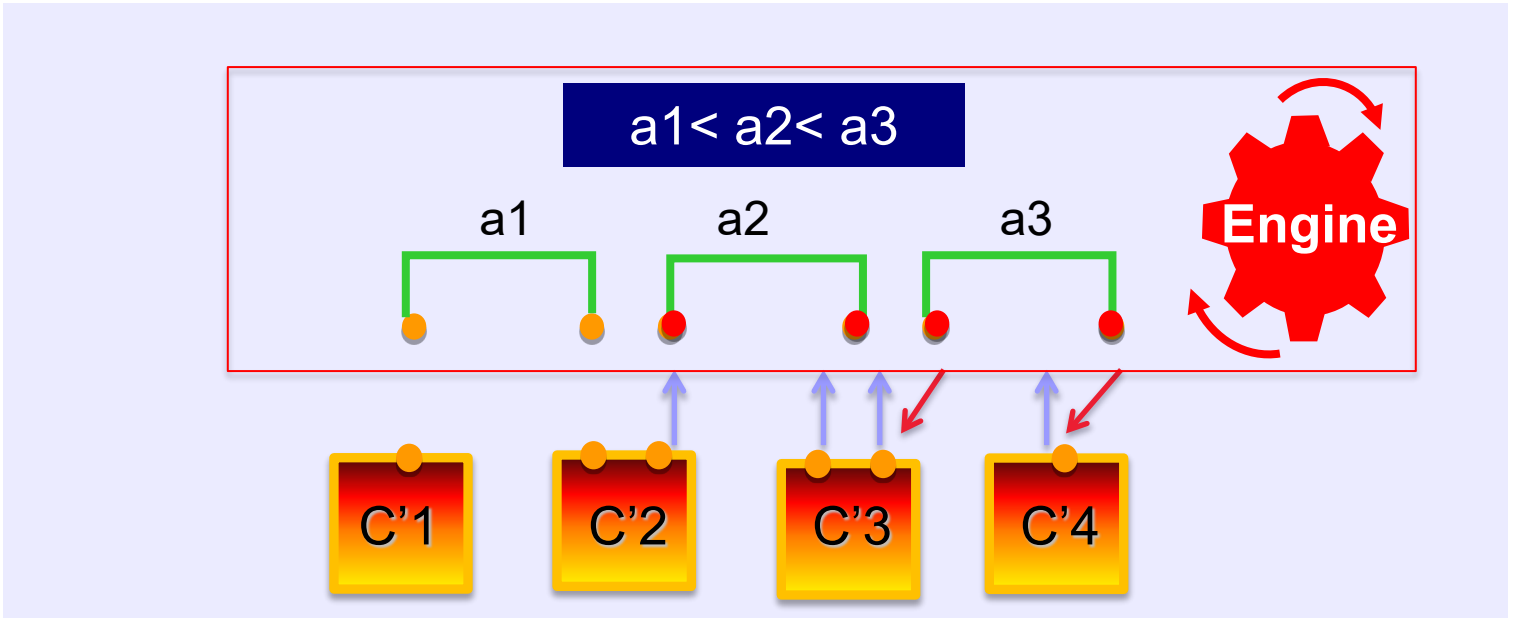
Partial state model

- Before reaching a ready state, the set of the enabled ports is sent to the Engine
- From a ready state, await notification from the Engine indicating the selected port

Distributed Implementation – Single Engine

BIP Model

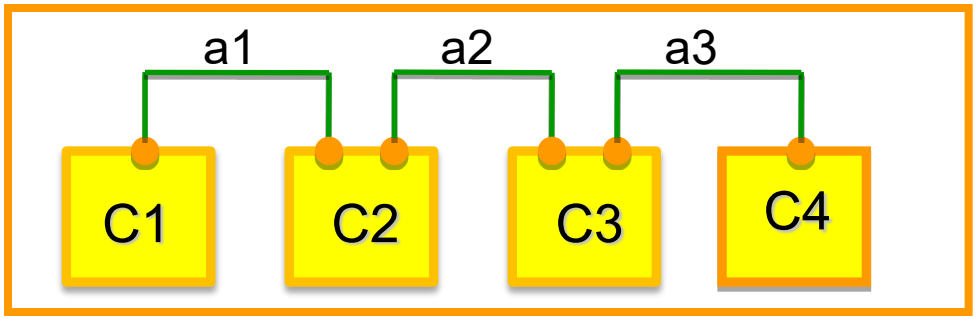


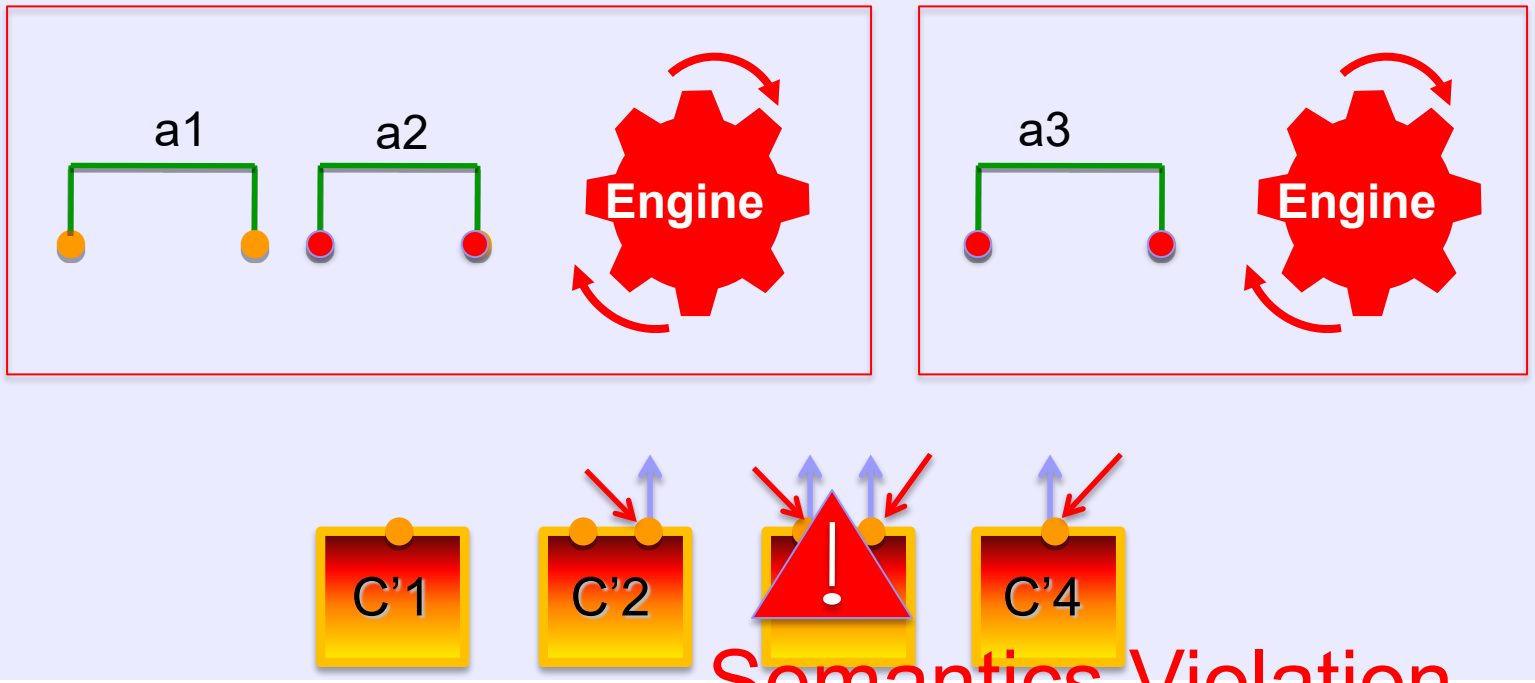


One Engine executes *all* interactions !

Distributed Implementation – Multiple Engines

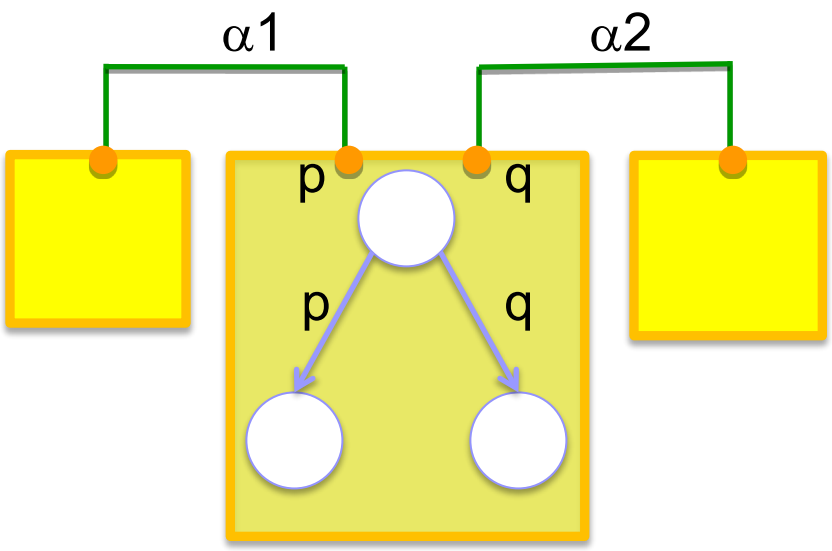
BIP Model



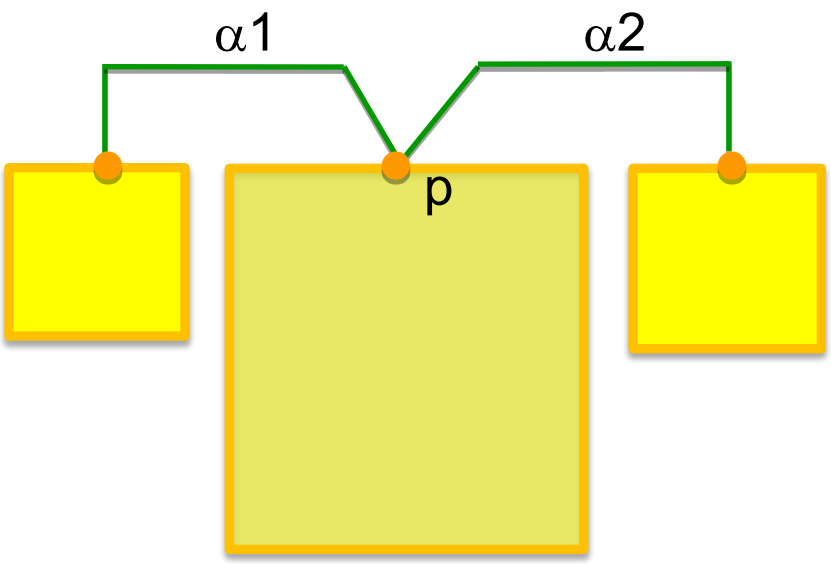


Dispatch interactions across *multiple* engines !

Distributed Implementation – Conflicting Interactions



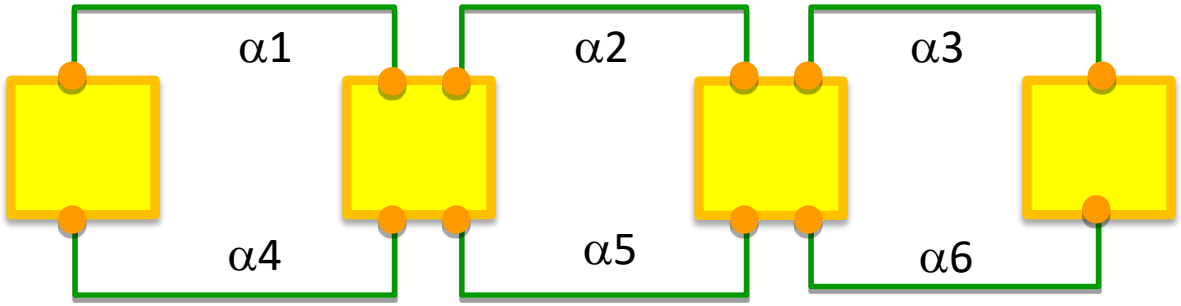
α1 and α2 depend on conflicting transitions



α1 and α2 share a common port p

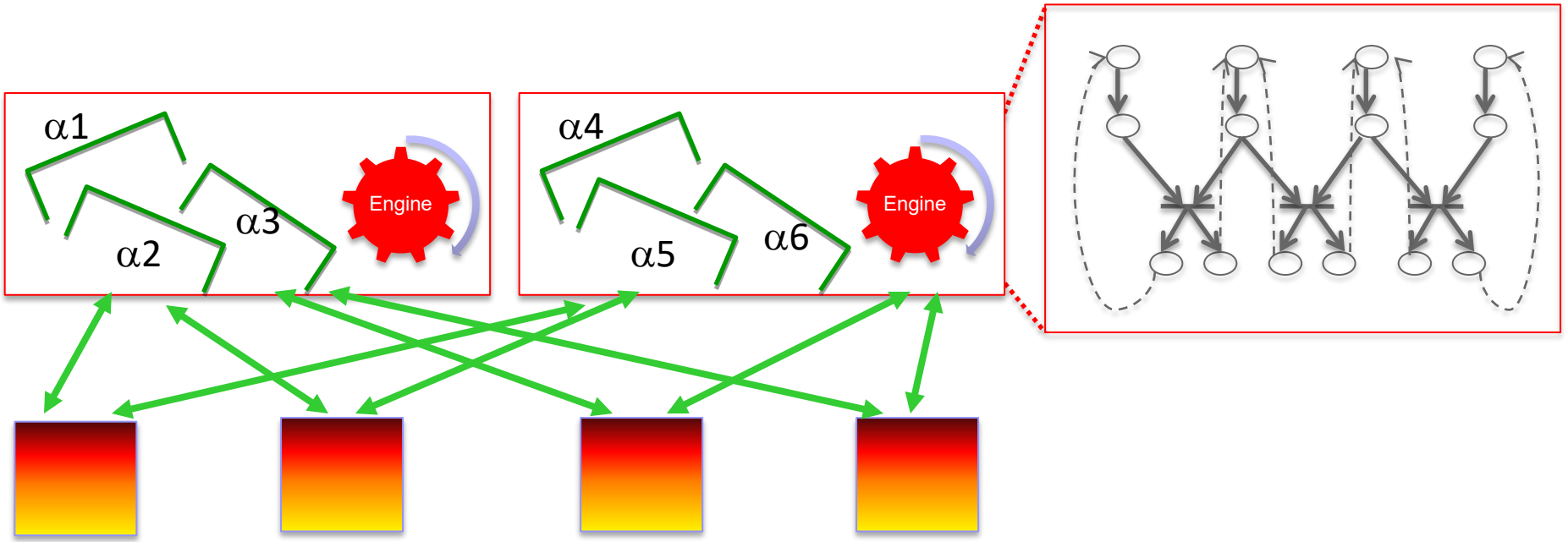
α1 # α2: the interactions α1 and α2 may be in conflict

Distributed Implementation – Conflict-Free Multiple Engines



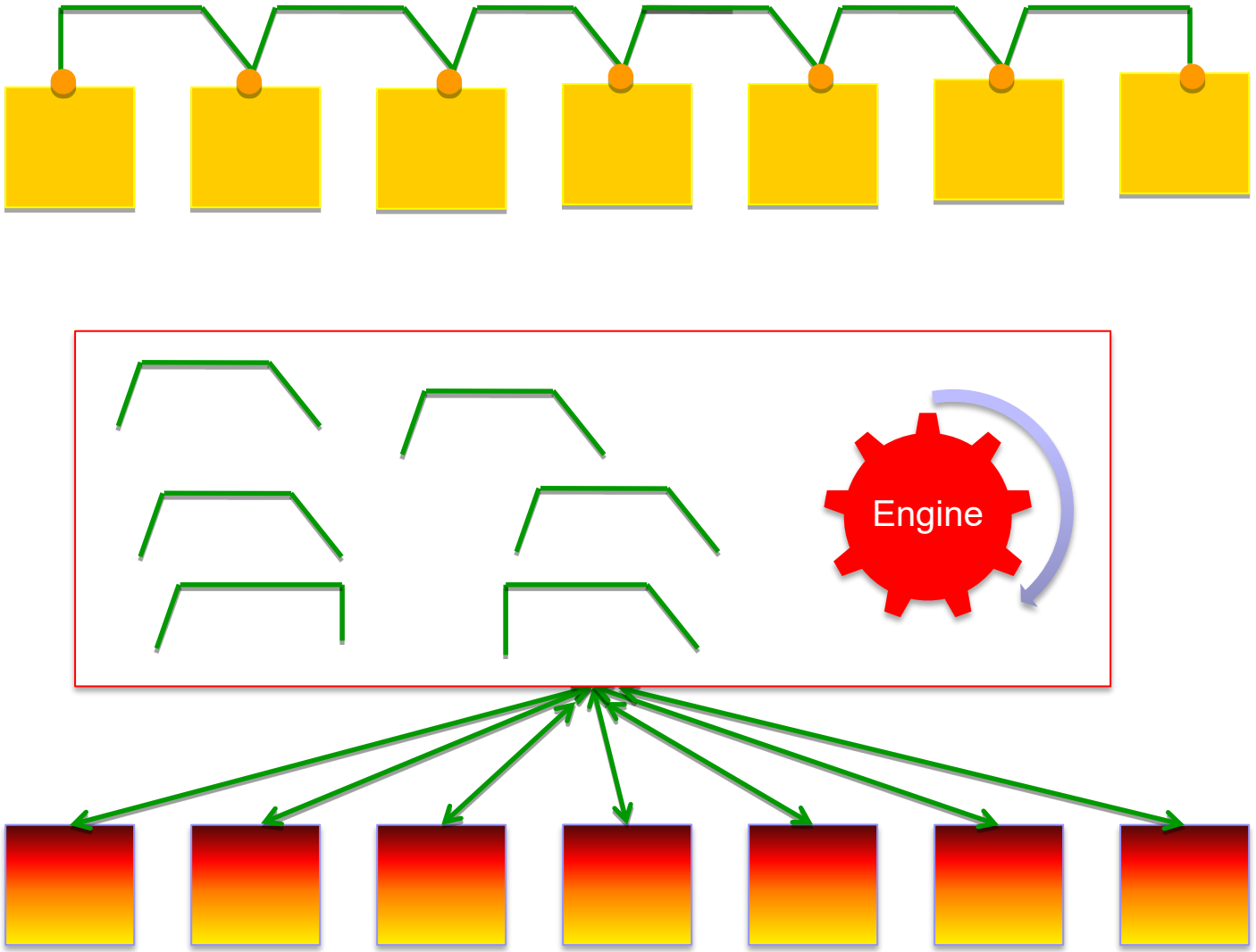
α1 # α2 # α3

α4 # α5 # α6



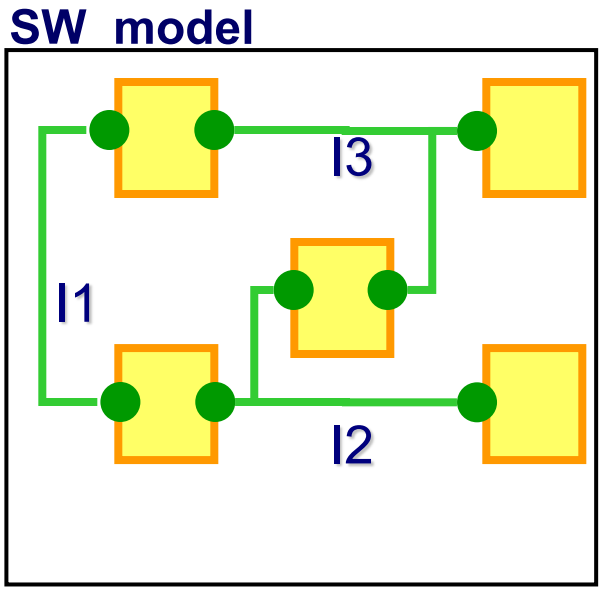
- Each engine handles interactions of a class of #*

Distributed Implementation – Limitations

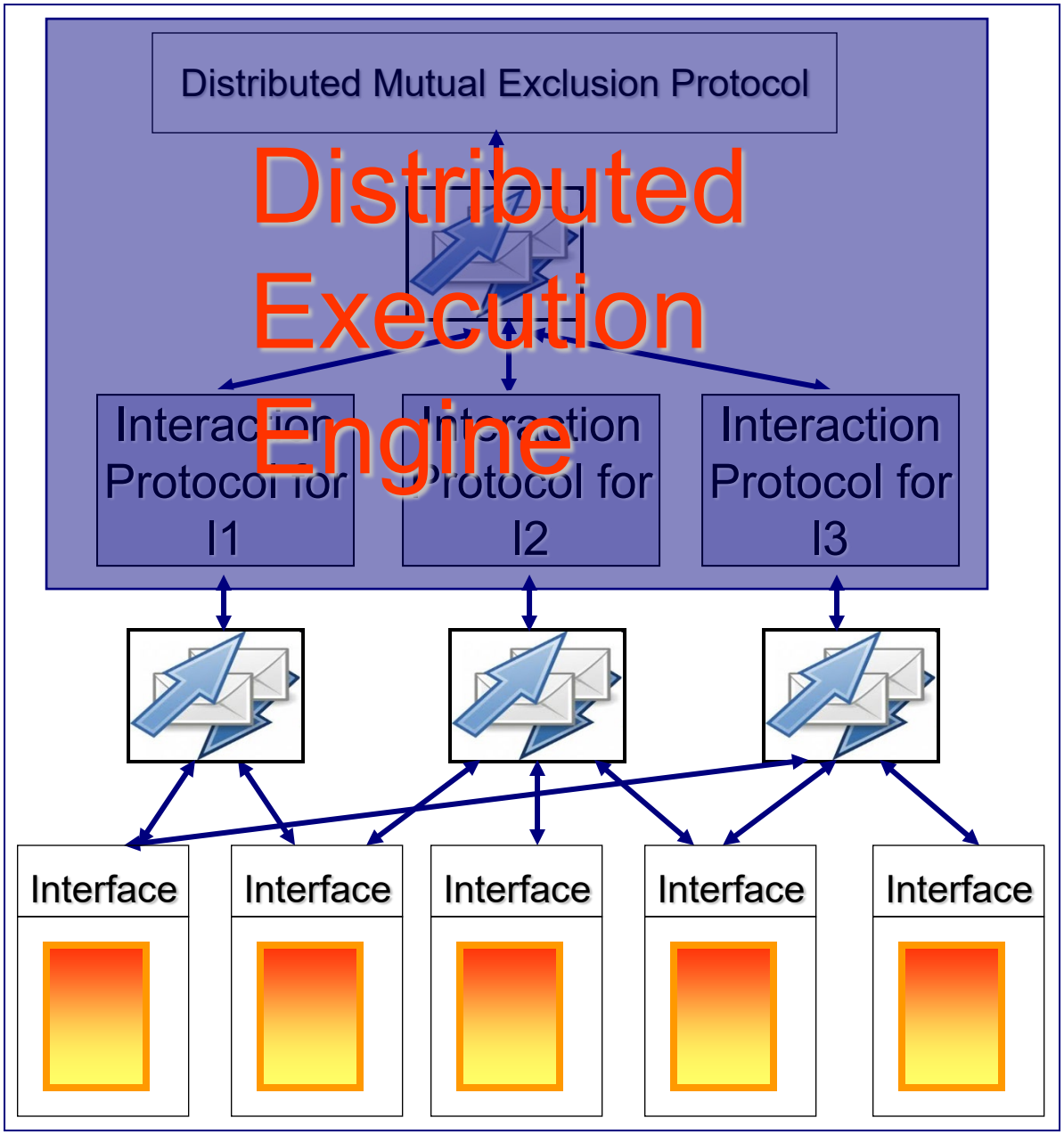


- Taking #* may reduce drastically parallelism between interactions

Distributed Implementation – 3-Layer Architecture

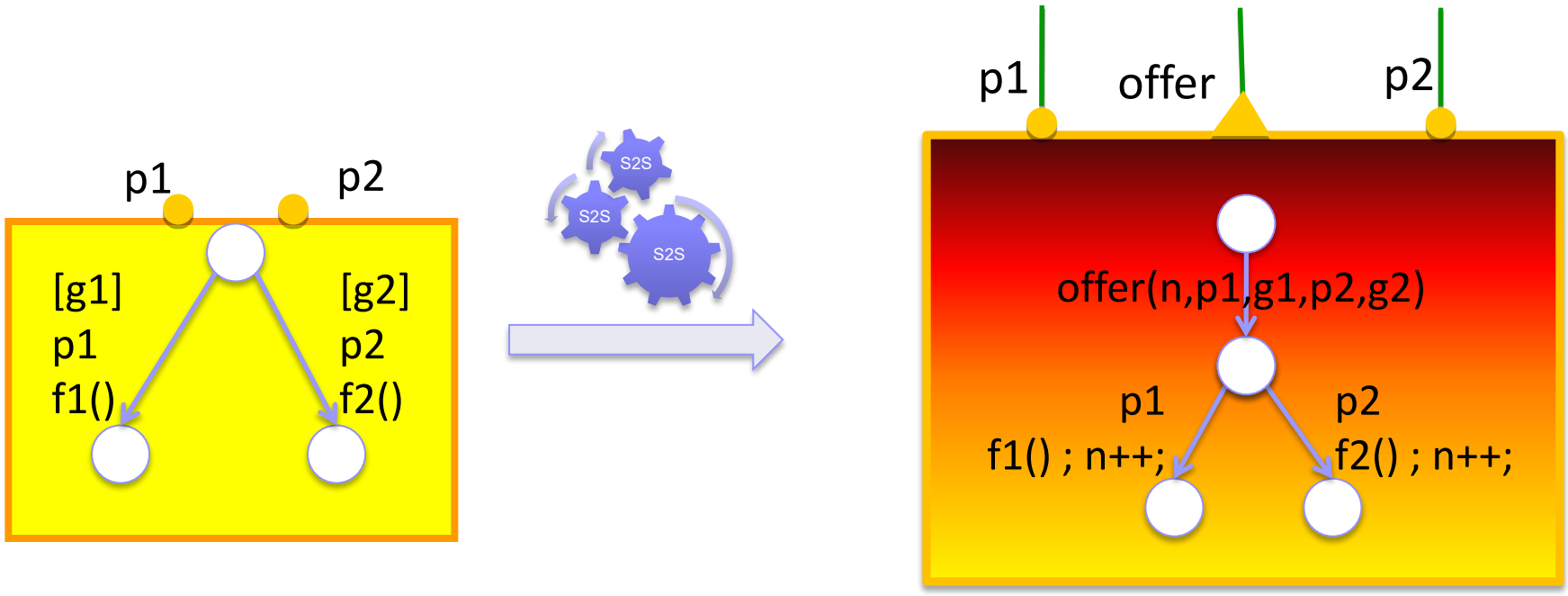


Distributed Implementation



Distributed Implementation – 3-Layer Architecture

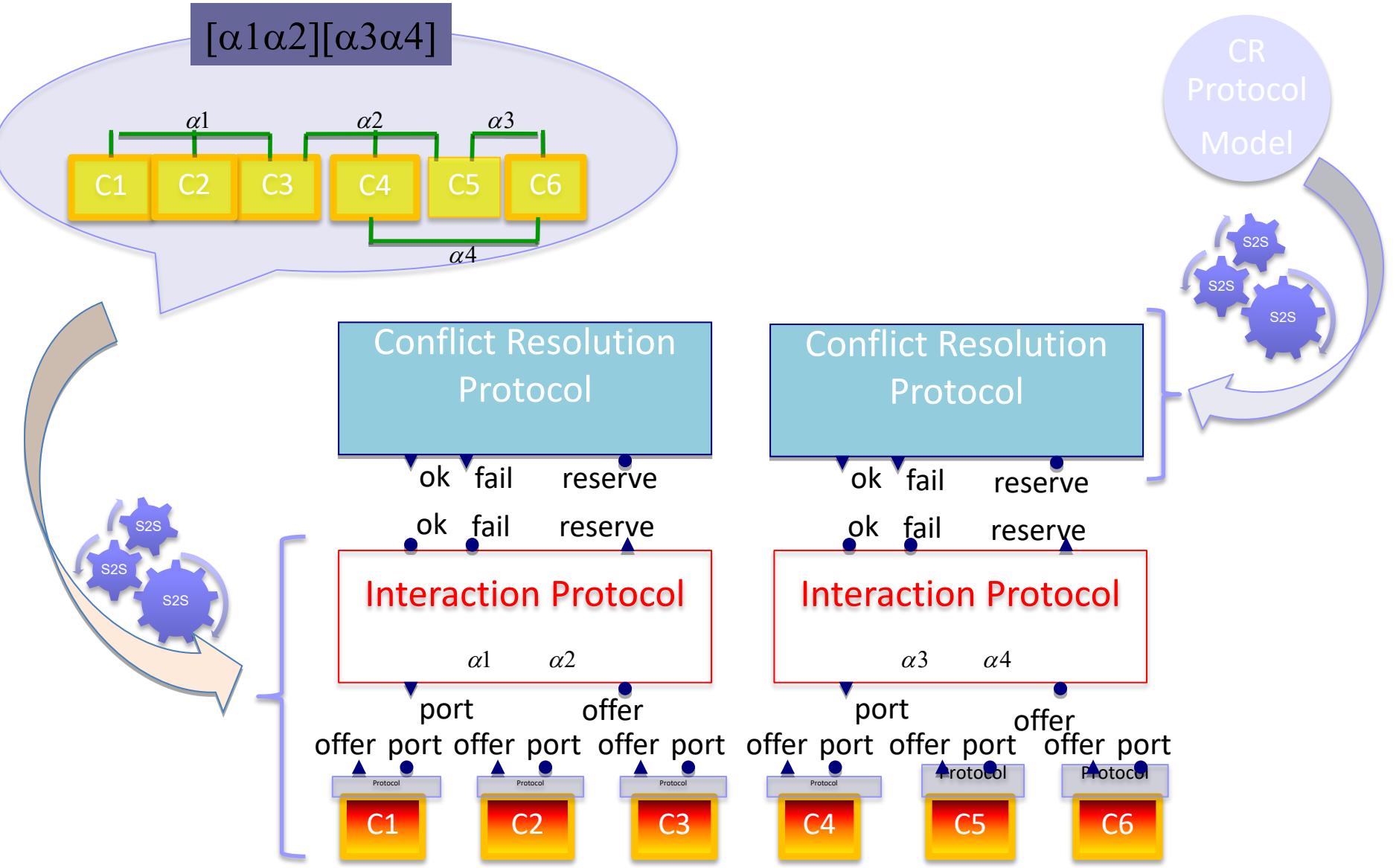
Transformation of atomic components



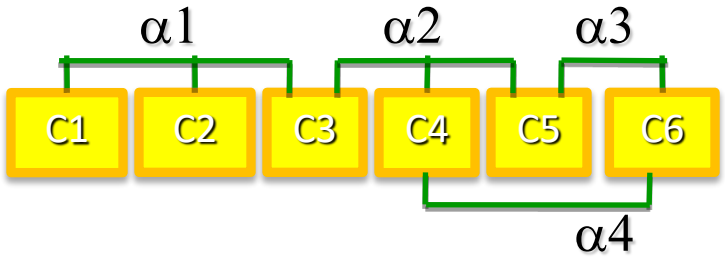
n: counts the number
of executed transitions

R. Bagrodia. A distributed algorithm to implement nparty rendevouz, FSTTCS 1987

Distributed Implementation – 3-Layer Architecture



Distributed Implementation – 3-Layer Architecture



Centralized Conflict Resolution Protocol

reserve2 ok2 fail2 reserve3 ok3 fail3 reserve4 ok4 fail4

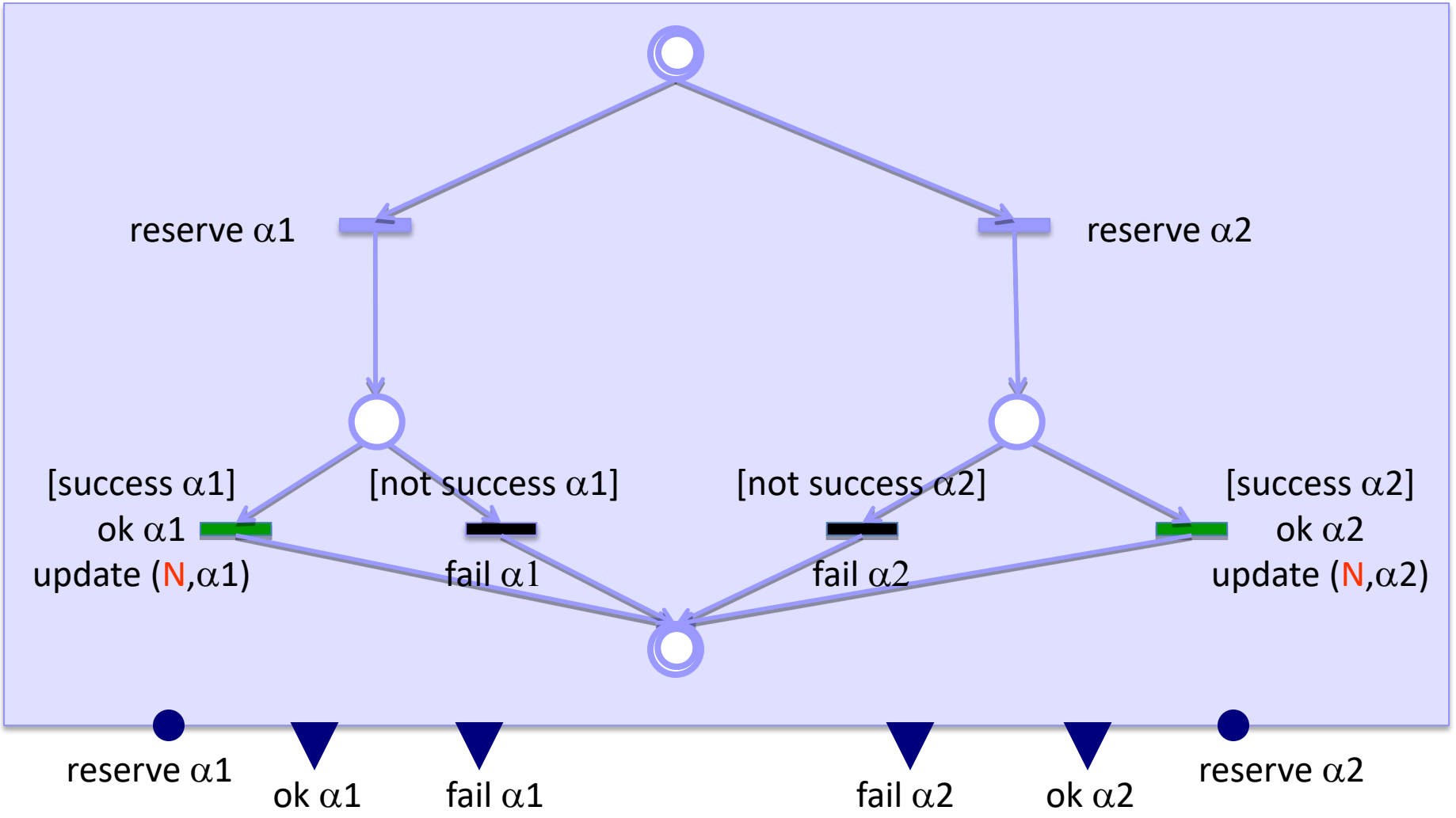
$\alpha 1$ Interaction Protocol $\alpha 2$

$\alpha 3$ Interaction Protocol $\alpha 4$

o1 p1 o2 p2 o3 p3 p4 o41 o42 p5 p6 o51 o52 p7 p8 o6 p5 p6

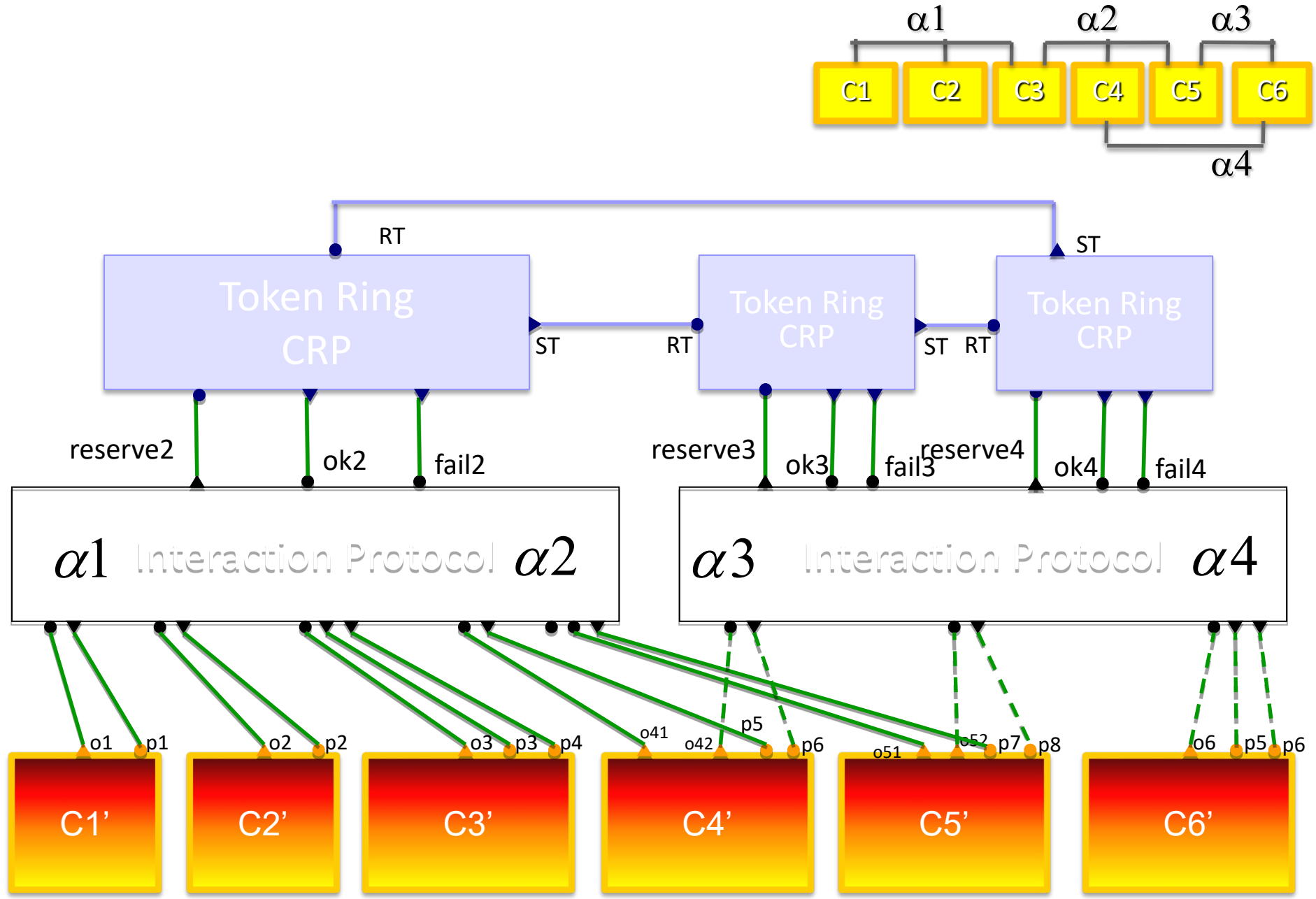
C1' C2' C3' C4' C5' C6'

Distributed Implementation – Centralized CRP

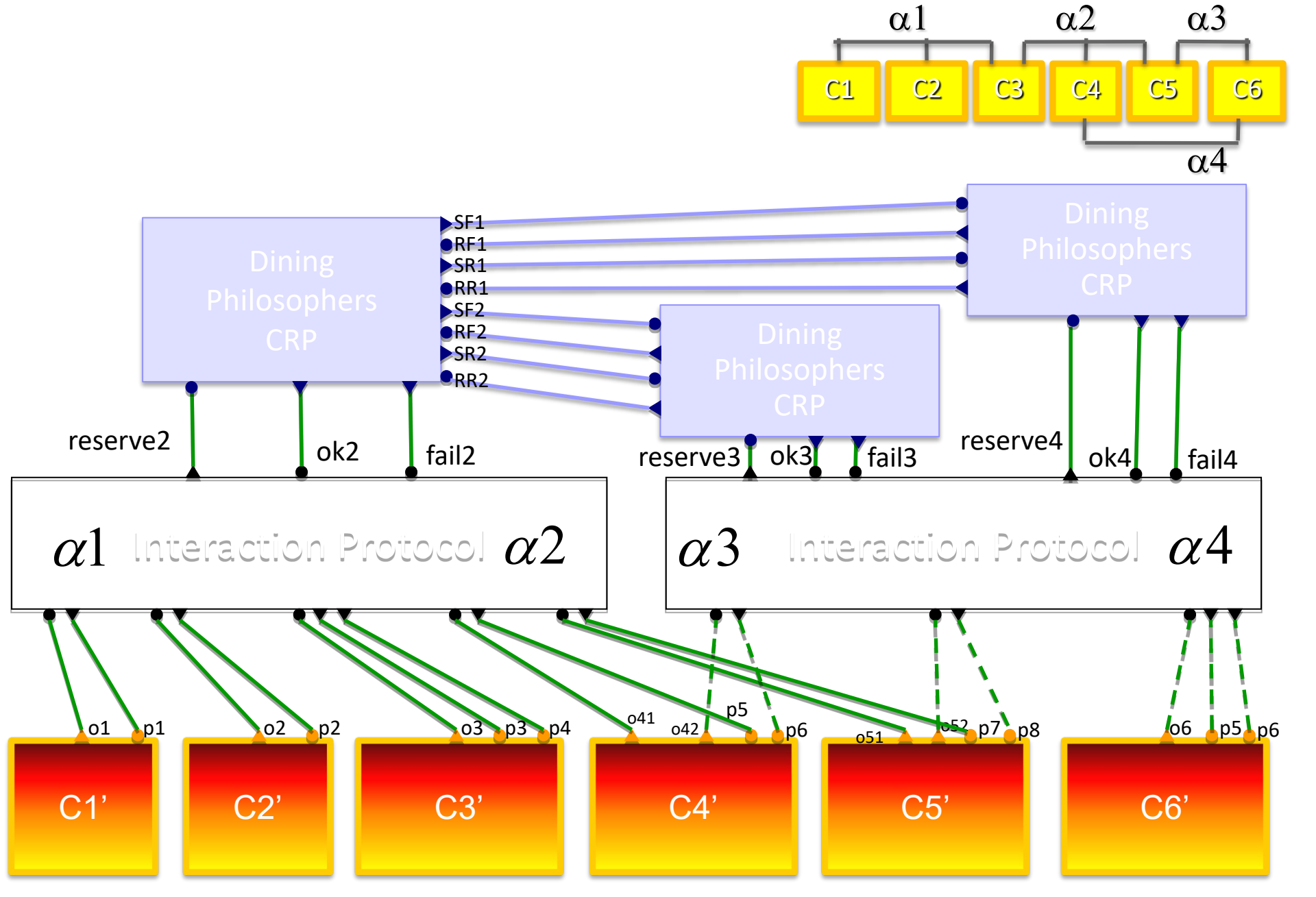


N=[N1, …, Nk]: keeps track of the state of the counters n of the components

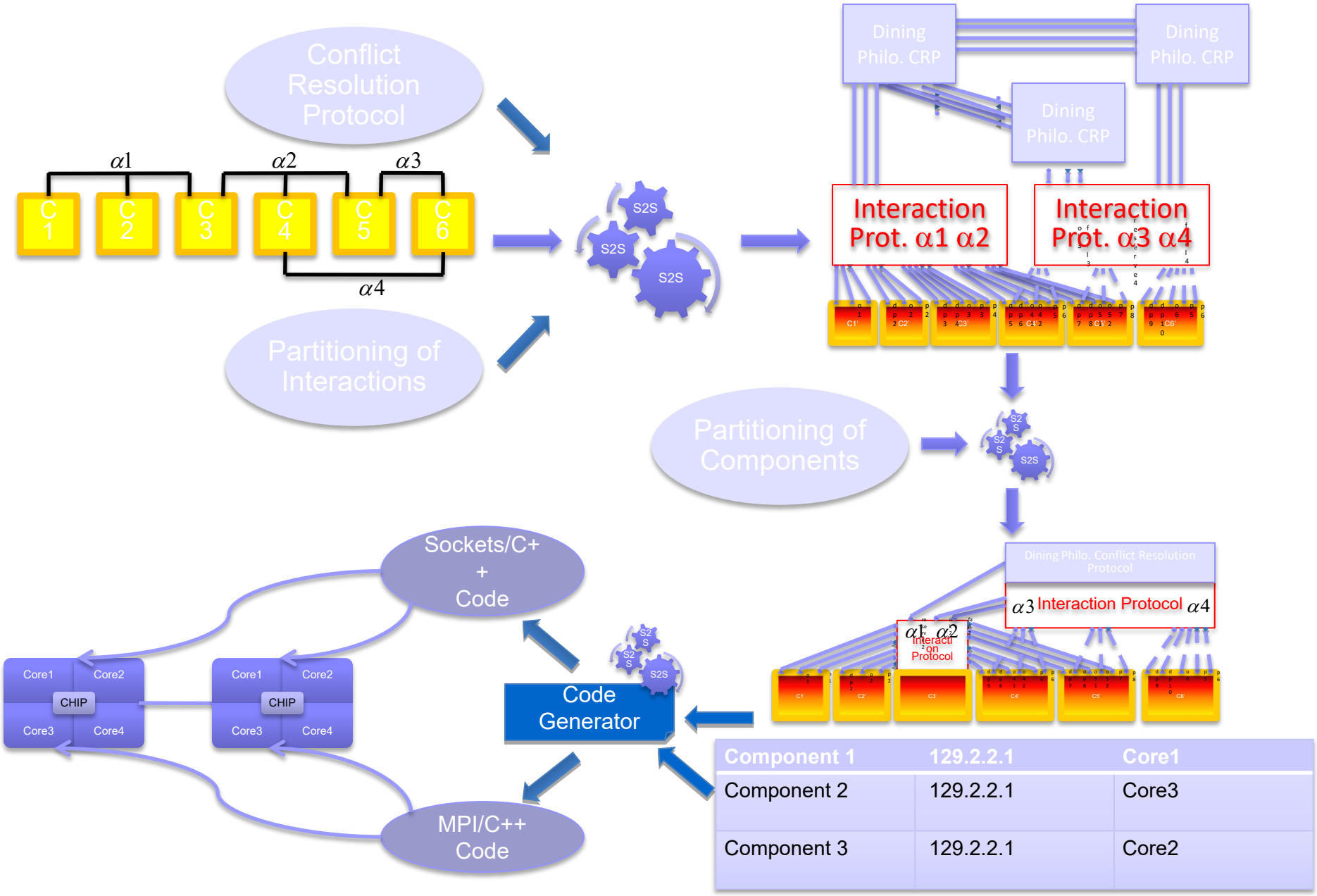
Distributed Implementation – Token Ring CRP



Distributed Implementation – Dining Philosophers CRP

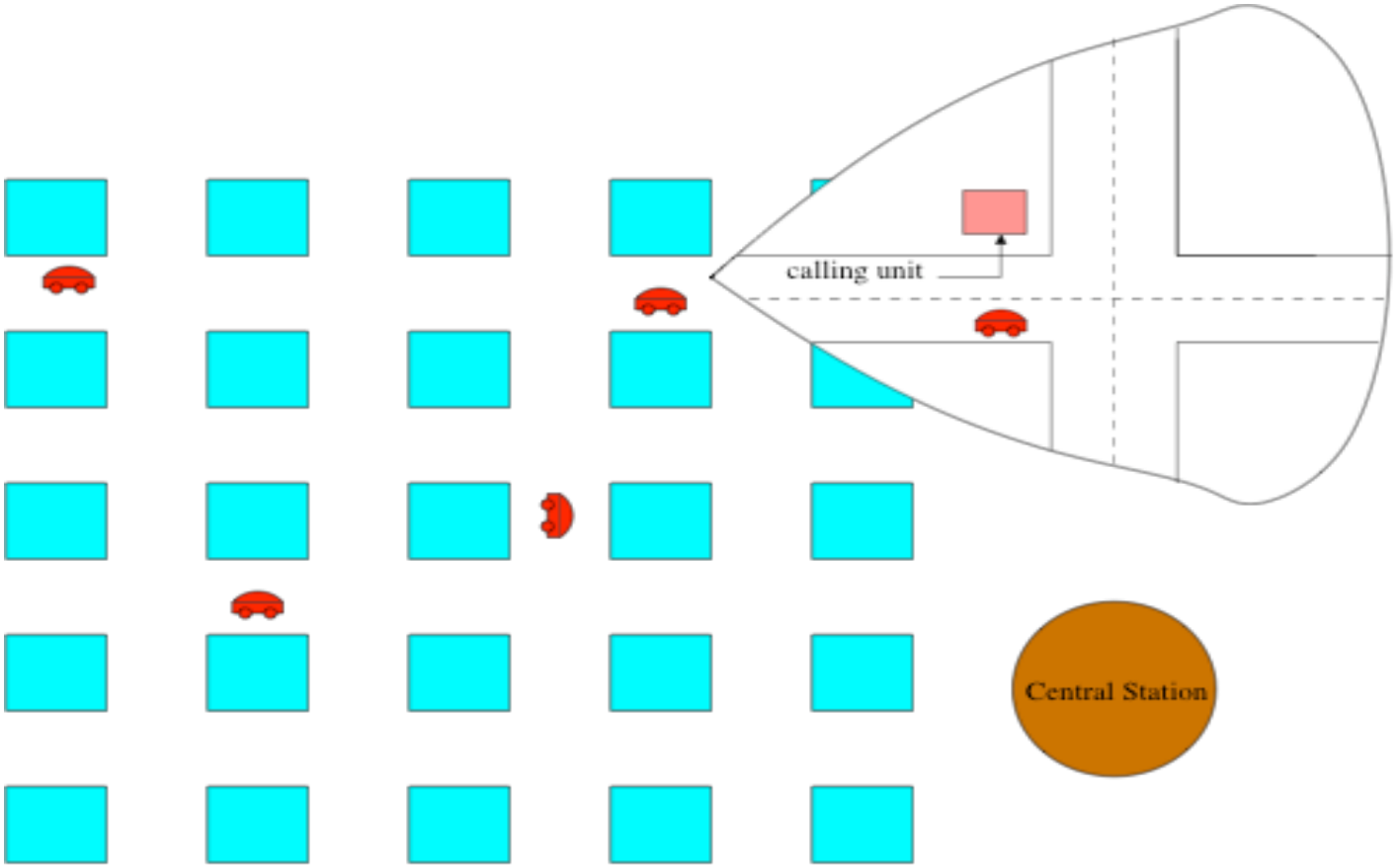


Distributed Implementation – Design Flow

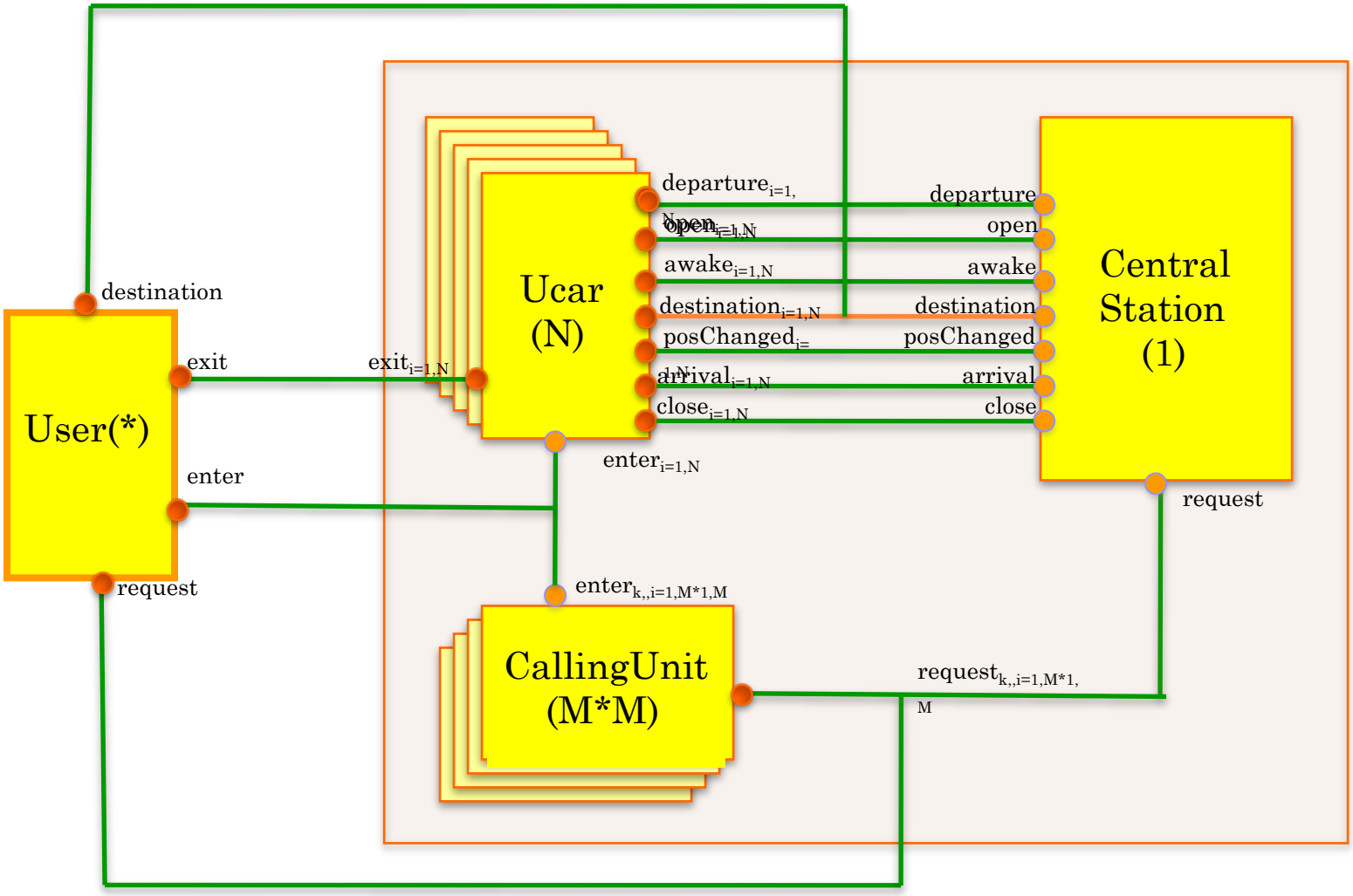


Distributed Implementation – Example

The UTOPAR system is an automated transportation system proposed by Israel Aircraft Industries in the COMBEST EU project



Distributed Implementation – Example

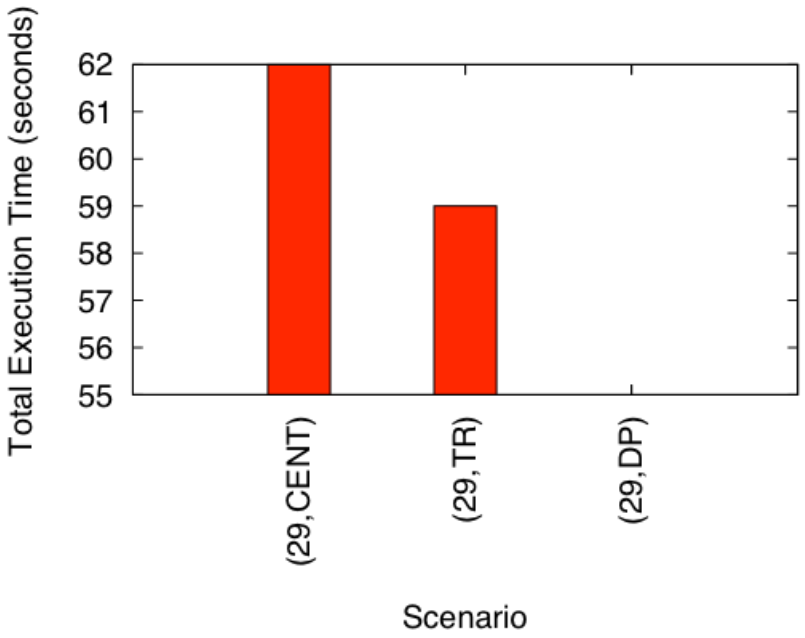


UTOPAR model in BIP

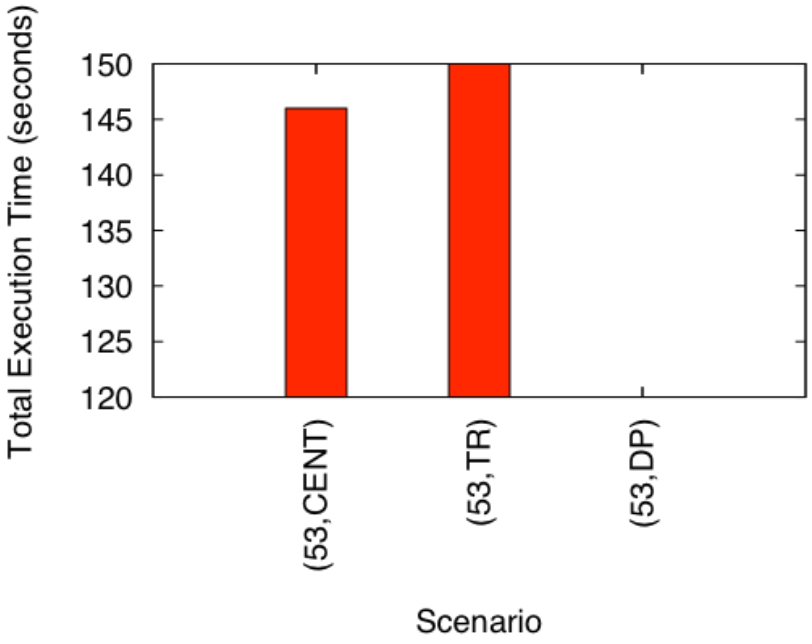
Distributed Implementation – Example

Benchmarks for fully automated generation of distributed C++ code for Linux sockets

25 = 5 * 5 calling units and 4 cars –
29 Interaction Protocols



49 = 7 * 7 calling units and 4 cars –
53 Interaction Protocols



Distributed Implementation – Further Developments

Priorities:

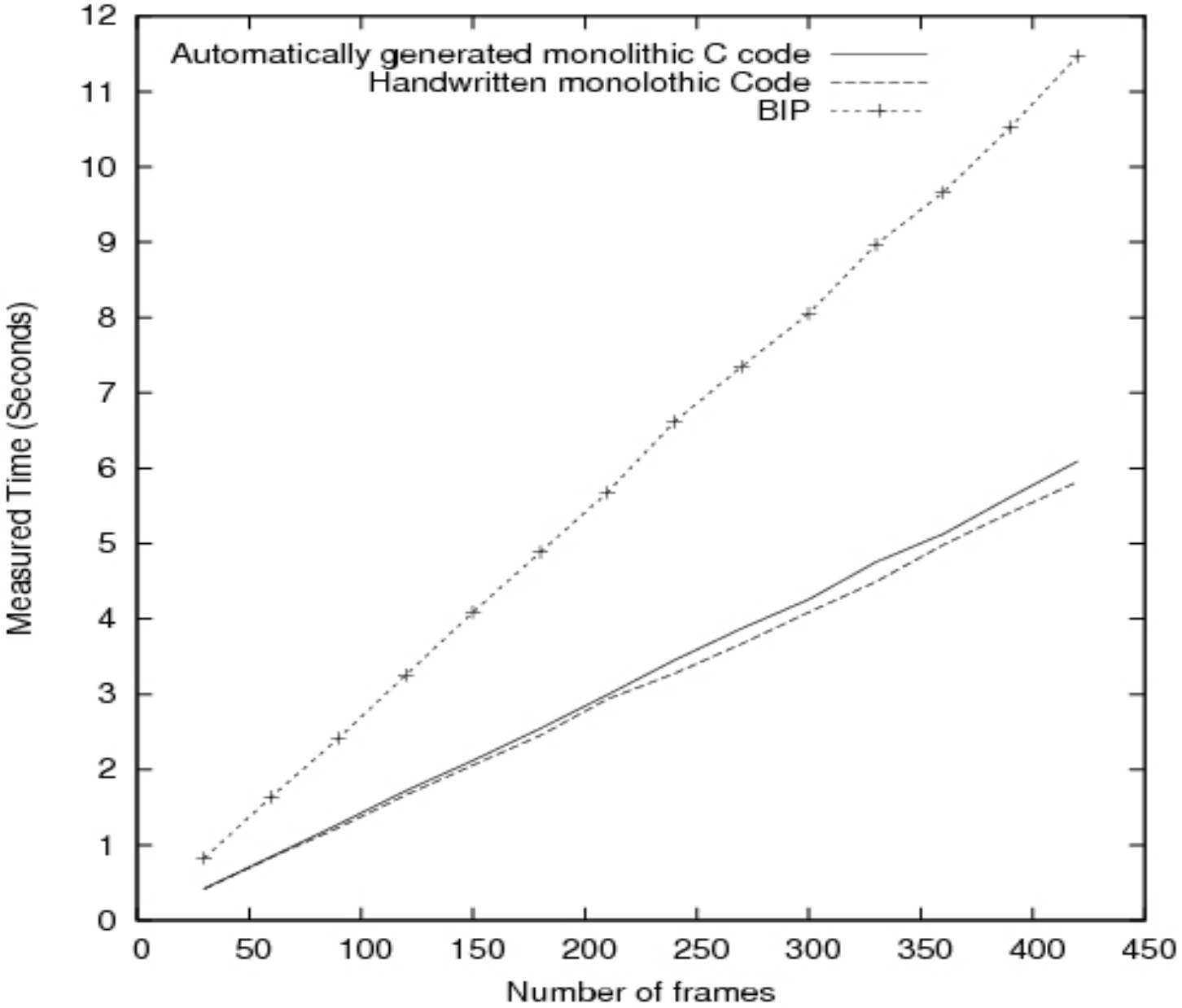
BIP model with priorities can be transformed into an equivalent model without priorities. The same implementation principle can be applied

Optimization issues:

- Building a correct snapshot of the system state is possible, but induces a lot of communication
  - Knowledge-based optimization by detecting false conflicts
  - Optimizing observability of interaction protocols (for priorities)

- Code optimization for components implemented on the same site
  - Replace a composite component by a single flattened component from which sequential monolithic code can be generated

Distributed Implementation – Monolithic Code Generation



- System Design
- Rigorous System Design
  - Separation of Concerns
  - Component-based Design
  - Semantically Coherent Design
  - Correct-by-construction Design
- The BIP Framework
  - Modeling in BIP
  - The BIP toolset
  - Compositional Verification
  - Synchronous Systems
  - SW Componentization
  - HW-driven Refinement
  - Distributed Implementation
- Discussion

OVERVIEW

Discussion – Breaking with Old Ideas

- New trends break with traditional Computing Systems Engineering - Goodbye to desktop applications and their ilk

- Too much of research in software engineering, systems, formal methods, etc. never made it in practice because it assumed a "design from scratch" approach and correctness-by-checking

- Formal methods
  - can only partially contribute to enhancing trustworthiness and optimality
  - are limited to systems and properties that can be formalized and checked efficiently e.g. functional properties of SW components

Discussion – Rigorous System Design

New approach for the design of trustworthy and optimized systems

❑ Endeavors unification through formalization of design as a process

- for deriving trustworthy and optimal implementations from an application software and models of its execution platform and physical environment
- which is semantically sound, incremental, scalable and accountable

❑Opens the way for moving from ad hoc and empirical design techniques to a well-founded design discipline

Discussion – About BIP

The BIP component framework has been developed for more than 10 years, with Rigorous Design in mind

- Translation of DSL (Simulink, Lustre, DOL, nesC) into BIP
- Source-to-source transformations proven correct-by-construction
  - taking into account HW resources
  - generating distributed implementations for several platforms
  - code optimization
- Run-times for centralized execution/simulation, distributed execution, real-time execution
- Validation and analysis tools
  - Incremental checking for Deadlock-freedom: D-Finder tool
  - Statistical Model Checking
- Successful application in many industrial projects
  - software componentization for robotic systems (DALA Space Robot for Astrium)
  - programming multi-core systems (P2012 for STM, MPPA for Kalray)
  - complex systems modeling (AFDX and IMA for Airbus)

Discussion – A System-Centric Vision for CS

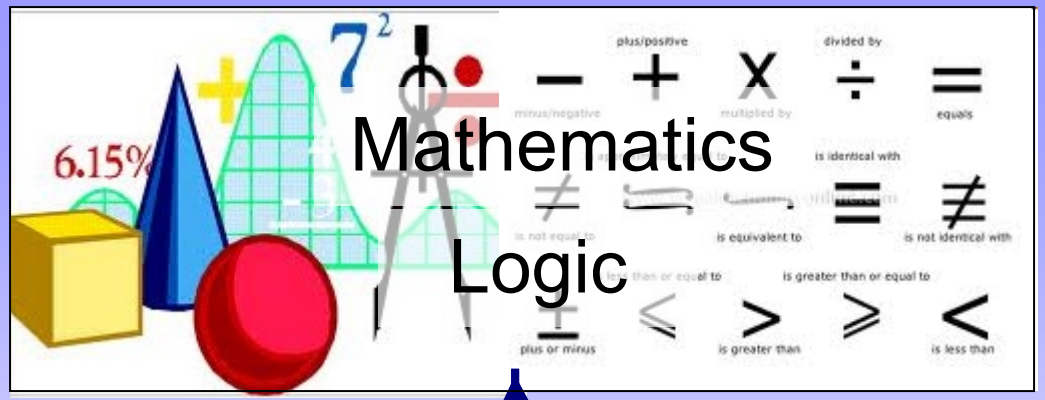
System Design

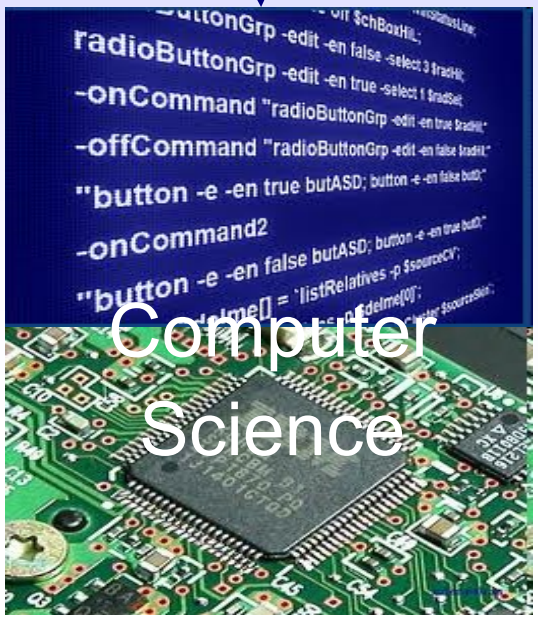
- Raises a multitude of deep theoretical problems such as the conceptualization of needs in a given area and their effective transformation into correct artifacts.

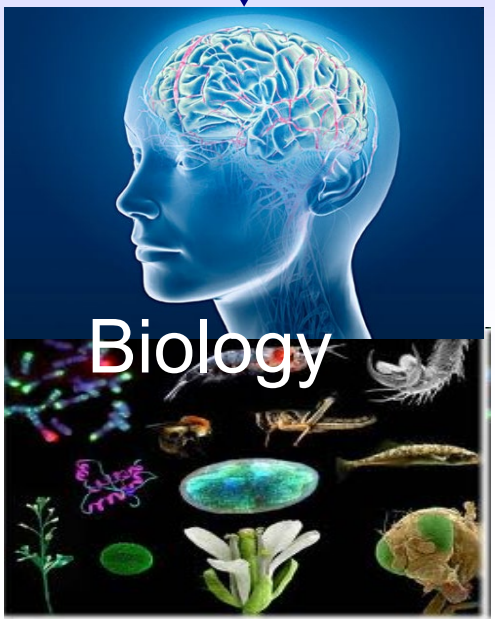
- has attracted little attention from scientific communities and is relegated to second class status
  - design is by its nature multi-disciplinary and requires consistent integration of heterogeneous system models supporting different levels of abstraction including logics, algorithms and programs as well as physical system models.

- is central to CS. Awareness on its centrality is a chance to reinvigorate CS research and build new scientific foundations matching the needs for increasing system integration and new applications

Discussion – A System-Centric Vision for CS: The Frontiers







Discussion – A System-Centric Vision for CS: The Frontiers

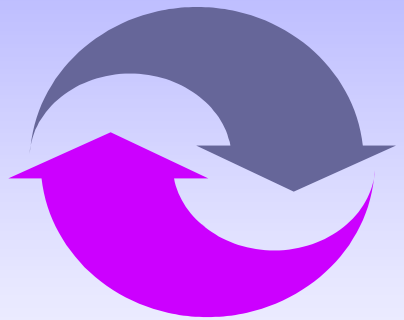
The Physical Hierarchy	The Cyber-Hierarchy	The Bio-Hierarchy
The Universe	The Cyber-world	Ecosystem
Galaxy	Networked System	Organism
Solar System	Reactive System	Organ
Electro-mechanical System	Virtual Machine	Tissue
Crystals-Fluids-Gases	Instruction Set Architecture	Cell
Molecules	Integrated Circuit	Protein and RNA networks
Atoms	Logical Gate	Protein and RNA
Particles	Transistor	Genes

We need theory, methods and tools for climbing up-and-down the abstraction hierarchy

Discussion – A System-Centric Vision for CS: The Frontiers

Physics

- Deals with phenomena of the « real » physical world (transformations of matter and energy)
- Focuses mainly on the discovery of physical laws.
- Physical systems – Analytic models
- Continuous mathematics
- Differential equations - robustness
- Predictability for classical Physics
- Mature discipline



Computer Sc.

- Deals with the representation and transformation information
- Focuses mainly on the construction of systems
- Computing systems – Machines
- Discrete mathematics – Logic
- Automata, Algorithms, Complexity Theory
- Verification, Testing,
- Young fast evolving discipline

Discussion – A System-Centric Vision for CS: The Frontiers

Artificial vs. Natural Intelligence

Living organisms intimately combine interacting physical and computational phenomena that have a deep impact on their development and evolution

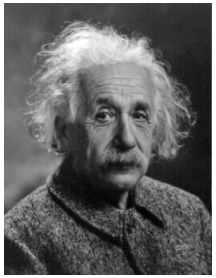
- Shared characteristics with computing systems
  - use of memory
  - distinction between hardware and software
  - use of languages
- Remarkable differences :
  - robustness of computation
  - built-in mechanisms for adaptivity
  - emergence of abstractions – concepts

Interactions and cross-fertilization

- Non von Neumann computing $\Leftarrow$ Neuromorphic, Cognitive Computing
- CAD methods&tools $\Rightarrow$ Synthetic Biology

Discussion – Looking for Foundations

Theoretical research in CS often focuses on "nice theory" that is not always practically relevant



"Make everything as simple as possible, but not simpler"

"….. in the academic world the theories that are more likely to attract a devoted following are those that best allow a clever but not very original young man to demonstrate his cleverness."

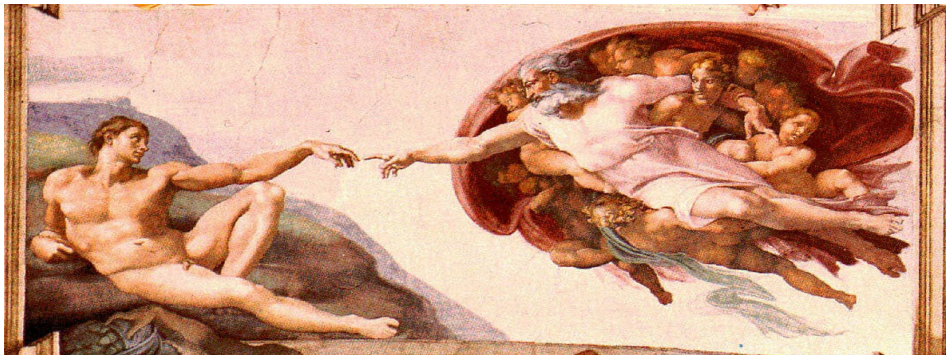
…. while practitioners propose ad hoc frameworks hardly amenable to formalization e.g. non-orthogonal concepts, ambiguous semantics

"Perfection is reached not when there is no longer anything to add, but when there is no longer anything to take away"

Discussion – Looking for Foundations

Is it possible to find a mathematically elegant and still practicable theoretical framework for system design?

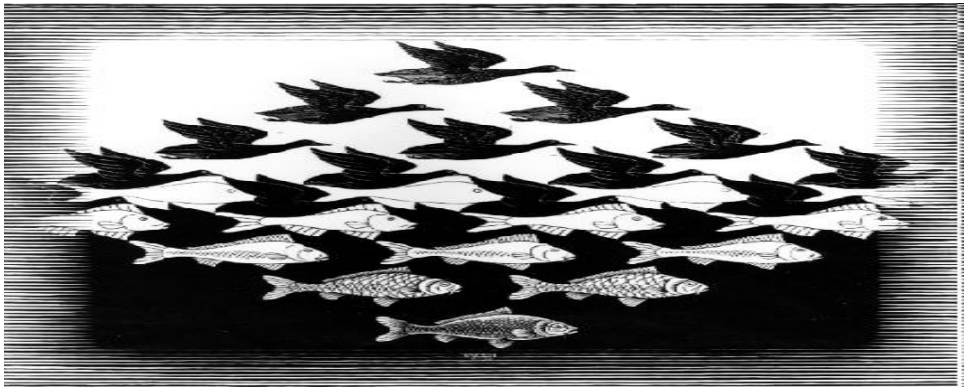
Physics and Biology study a given "reality"



The key issue is discovering laws governing phenomena

"The most incomprehensible thing about the world is that it is at all comprehensible."

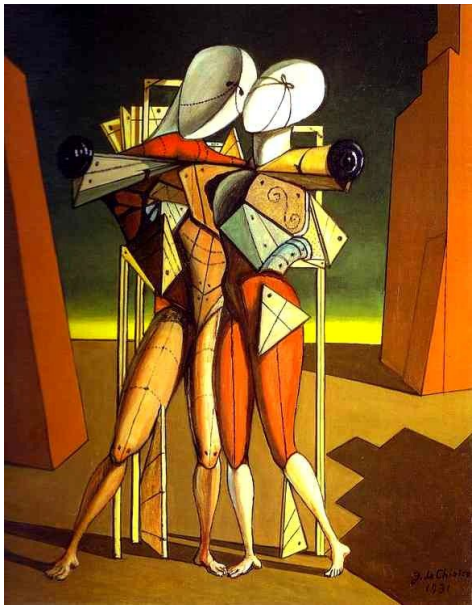
Computer Science mainly deals with building systems (artifacts)



The key issue is building
correct systems
cost-effectively

Discussion – Looking for Foundations

The physical world is part of our conditions of existence



We mold the conditions of existence of the cyber-world

Is everything for the best in the best of all possible cyber-worlds ?

Publications

Saddek Bensalem, Marius Bozga, Joseph Sifakis, Thanh-Hung Nguyen. "Compositional Verification for Component-Based Systems and Application". ATVA 2008: 64-79

Simon Bliudze, J. Sifakis. A Notion of Glue Expressiveness for Component-Based Systems. In Proc. of the 19th International Conference on Concurrency Theory (CONCUR'08), LNCS 5201, 508–522, Springer, 2008.

Ananda Basu, Matthieu Gallien, Charles Lesire, Thanh-Hung Nguyen, Saddek Bensalem, Felix Ingrand and Joseph Sifakis. "Incremental Component-Based Construction and Verification of a Robotic System." ECAI 2008 The 18th European Conference on Artificial Intelligence, Patras, Greece, July 21 - 25, 2008.

Simon Bliudze, J. Sifakis. The Algeba of Connectors—Structuring Interaction in BIP. IEEE Transactions on Computers, vol. 57, no. 10, pp. 1315–1330, October, 2008.

A. Basu, Ph. Bidinger, M. Bozga and Joseph Sifakis. Distributed Semantics and Implementation for Systems with Interaction and Priority FORTE, 2008, pp. 116-133.

T.A. Henzinger and J. Sifakis. The Discipline of Embedded Systems Design Computer, October 2007, pp. 32-40.

S. Bliudze, J. Sifakis. The Algebra of Connectors – Structuring Interaction in BIP Proc. EmSoft07, ACM&IEEE, Oct. 1-3, 2007, Salzburg, Austria, pp. 11-20.

Publications (2)

A. Basu, L. Mounier, M. Poulhiès, J. Pulou and J. Sifakis. Using BIP for Modeling and Verification of Networked Systems - A Case Study on TinyOS-based Networks Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), 12 - 14 July 2007, Cambridge, MA, USA, pages 257-260.

G. Gössler, S. Graf, M. Majster-Cederbaum, M. Martens, J. Sifakis. An Approach to Modeling and Verification of Component Based Systems in Current Trends in Theory and Practice of Computer Science, SOFSEM'07, LNCS 4362, 2007.

G. Gössler, S. Graf, M. Majster-Cederbaum, M. Martens, J. Sifakis. Ensuring Properties of Interaction Systems by Construction,Program Analysis and Compilation, Theory and Practice, LNCS, 2007.

A. Basu, M. Bozga, J. Sifakis. Modeling Heterogeneous Real-time Systems in BIP 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM06),Invited talk, September 11-15, 2006, Pune, pp. 3-12.

M. Poulhiès, J. Pulou, C. Rippert and J. Sifakis. A Methodology and Supporting Tools for the Development of Component-Based Embedded Systems, 13th Monterey Workshop 2006, Paris, October 2006, pp. 75-96, LNCS 4888

T.A. Henzinger and J. Sifakis. The Embedded Systems Design Challenge Invited Paper, FM 2006, pp. 1-15.

Thank You